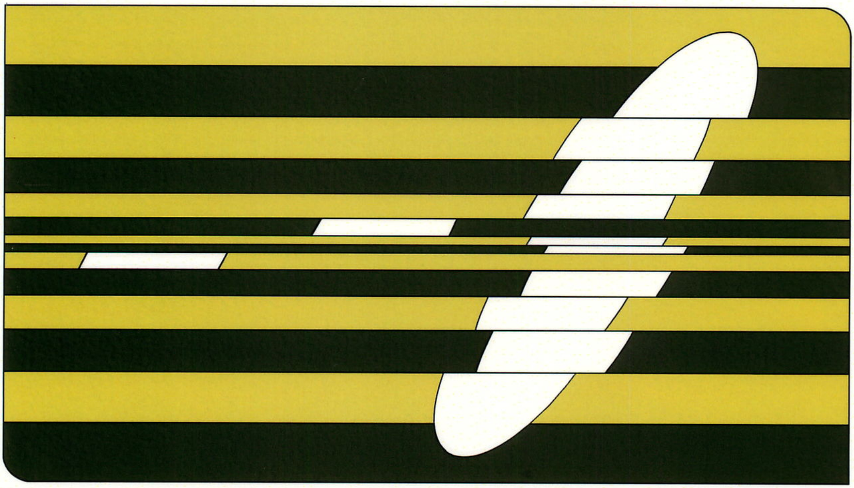
編集委員：武市正人
米澤明憲

レクチャーノート/ソフトウェア学

1

型理論

龍田　真著



近代科学社

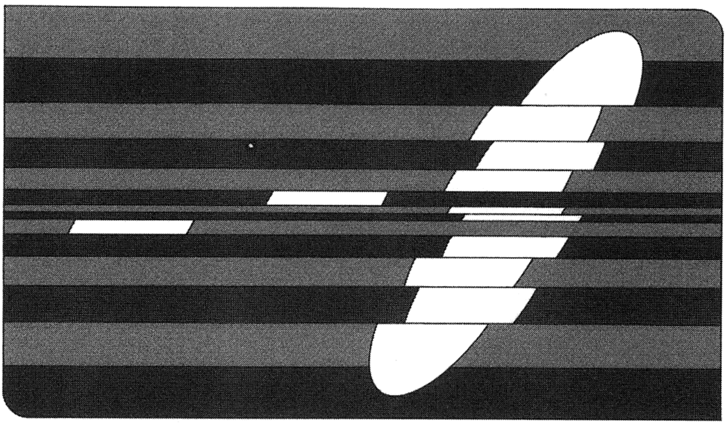
日本ソフトウェア科学会

レクチャーノート/ソフトウェア学

1

型理論

龍田　真著



近代科学社

日本ソフトウェア科学会

はじめに

型理論について概説する．様々な型理論が提案されているが，ここでは，基本的で理解しやすい単純型理論 $\lambda\!\rightarrow$, すでに実用的プログラム言語に取り入れられて使われている ML の型推論，predicative な型理論の代表である $\mathbf{ITT}_n$, impredicative な型理論の代表の一つである F, impredicative な型理論の中で最も強力なものの一つである CC の五つの基本的で代表的な型理論を紹介する．最後に，様々な型理論の関係を整理して理解するため，λ-cube と，Girard の逆説を紹介する．

各型理論について，具体的な理解のため，形式体系の完全な定義を紹介し，推論の例を説明する．また，型理論が計算機科学にどのように役立つかという観点に重点を置いて説明を行う．

1,2 章では，最も単純な型理論である $\lambda\!\rightarrow$ を紹介し，また，他の型理論にも一般に共通である基本的概念，性質について説明する．

3,4 章では，プログラム言語 ML に取り入れられている，最も実用的な型理論の一つである ML の型推論について説明する．ML における型推論は，強力な型の機構であるにもかかわらず，計算可能である点が最大の特色である．ML における型推論を型理論の形式的体系を用いて説明した後，型推論のアルゴリズムを紹介する．

5 章では，predicative な型理論の代表である Martin-Löf の型理論 $\mathbf{ITT}_n$ を説明する．$\mathbf{ITT}_n$ は，自然で明晰な概念の積み重ねから構成されている．$\mathbf{ITT}_n$ はたくさんの推論規則からなる型理論であるが，このことの具体的な理解のため，ここでは，$\mathbf{ITT}_0$ の完全な推論規則を紹介し，推論の例を説明する．また，論理体系の $\mathbf{ITT}_n$ による解釈について説明する．また，$\mathbf{ITT}_n$ をプログラミングシステムに応用した例を紹介する．

6 章では，impredicative な型理論の代表の一つである Girard の F を簡単に紹介する．

7 章では，impredicative な型理論の中で最も強力なものの一つである Coquand の CC について紹介する．CC が強力で，計算機科学に役立つ型理論であることを説明するため，論理記号の定義の例と，帰納的型の定義の例を説明する．

8 章では，λ→，F，CC などの様々な型理論の関係を整理して理解するため，Barendregt の λ-cube を紹介する.

9 章では，強力な型理論に限界を与える Girard の逆説について説明する.

10 章では，型理論に関する基本的な文献を紹介する.

この解説は，日本ソフトウェア科学会の学会誌コンピュータソフトウェアに掲載された解説記事 [44, 45, 46, 47] を転載の承認を得てまとめて本の形にしたものである. 内容は，4 回にわたって連載された解説記事を一つにまとめ，若干の誤りを訂正したものである.

この解説を近代科学社から出版する機会を与えてくださった東京大学の武市正人助教授に感謝します. この解説の元となる解説記事をコンピュータソフトウェアに書く機会を与えてくださった明治大学の疋田輝雄教授に感謝します. また，草稿を読んでいただき，親切で的確なコメントをくださった東京大学の米澤明憲教授と日立製作所基礎研究所の金藤栄孝研究員に感謝します. 最後に，型理論も含めて筆者の研究に日頃から刺激を与えてくださり，また，この解説の執筆を励ましてくださった東北大学の佐藤雅彦教授に深く感謝します.

もくじ

第 1 章

型理論とは何か

ここで考える型理論の型とは，プログラム言語におけるデータに対するデータの型とほぼ同じもので，それを一般化したものと考えてよい．

例えば，Pascal では，次のような変数宣言が用いられる．

```
var x, y : integer, a : real ;
```

この文は，変数 x,y が integer の型を持ち，変数 a が real の型を持つことを表している．また，関数宣言

```
function is_prime(x : integer) : boolean ;
```

は，関数 is_prime が，引数として integer をとり，返値として，boolean を返すことを宣言している．このとき，is_prime の型は，integer を定義域とし，boolean を値域とする関数の型 integer -> boolean であると考え，

```
is_prime : integer -> boolean
```

と表す．

型理論とは，型に関する性質を論じるための形式的体系である．その形式的体系により表される型付けの機構自体も型理論という．

型理論の一般の形を説明しよう．形式的体系としての型理論は，項 (term)，型 (type)，判定 (judgement)，推論規則により定義される．

項 は，型付けされる式である．項 t が型 A を持つことを，

$$t : A$$

または，

$$t \in A$$

で表す．

判定 は，型理論における基本的言明である．型理論により異なるが，

$x_1 : A_1, \ldots, x_n : A_n \vdash t : A$

A type

$a \in A$

$a = b \in A$

$A = B$

などが使われる．判定 $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$ は，$t : A$ 中に自由に現れる変数 $x_1, \ldots, x_n$ の型をそれぞれ $A_1, \ldots, A_n$ であると仮定すると t が型 A を持つことを意味する．

推論規則は，判定 を演繹するための規則である．推論規則を積み重ねて証明図を構成したとき，その結論となっている判定 は，この型理論で証明可能であるという．

以下では，式という言葉を意味のある文字列を広く指す場合にも用いる．

変数 $x_1, \ldots, x_n$, 型 $A_1, \ldots, A_n$ について，$x_1 : A_1, \ldots, x_n : A_n$ の形の式を 文脈 (context) という．文脈 は，式中に自由に現れる変数 x_i について，x_i が型 A_i を持つことを示す変数宣言の役割を果たす．

型理論の歴史を，計算機科学に関連する事項に重点をおいて，簡単に紹介しよう．

型理論は，元々，数理論理学の分野で研究されてきた．計算機科学の歴史よりも古くから，様々な型理論が研究されていた．1958 年には，Curry-Howard の同型対応が発見されている [25]．

1971 年に，Martin-Löf は，Curry-Howard の同型対応に基づいて，数学および論理学の基礎を与えるための型理論を構成した [31]．この型理論の最初のものは，自然で強力な impredicative な型理論であったが，矛盾していることを Girard が発見した．これを，Girard の逆説という．このときより，Martin-Löf は，正しいと自分が確信できる範囲内だけで，型理論を構成した．これが，構成的型理論 $\mathbf{ITT}_n$ である [33]．

$\mathbf{ITT}_n$ では，自然数全体の型 N などの正しいと確信できる型から，Π- 規則などの正しいと確信できる型から正しいと確信できる型を構成する推論規則を繰り返し用いて作られる型だけを扱う．このような型理論を，predicative な型理論という．これとは異なり，型上を動く変数の全称束縛を許す型など，始めから型全体が定まっているとして超越的な型の構成も許す型理論を，impredicative な型理論という．$\mathbf{ITT}_n$ は，predicative な型理論の代表的なものの一つであり，計算機科学に積極的に応用

されている.

1972 年に, Girard は, 高階論理のカット除去定理に関連して, impredicative で強力な型理論 F を構成した [15]. 後に, F は, 計算機科学における型理論の活発な研究の出発点となった.

計算機科学の分野では, 型変数を含む型である多相型の概念が, 1967 年に Strachey により提案され, 研究され始めた. 1974 年には, Reynolds が, Girard とは独立に, 多相型を扱うための F と等しい型理論を構成した [39]. ほぼ同様の型理論が, ほぼ時期を同じくして, 数理論理学の分野と, 計算機科学の両分野で発見されたことは興味深い.

多相型は, 1977 年, Gordon, Milner, Wadsworth による型付関数型言語 **ML** により, 本格的に実用化した [19]. **ML** の型の概念は, 型を持つプログラムの計算は必ず停止するという強正規性の数学的基礎を持ち, 多相型が使え, また, 型推論が行われる点で, Fortran や Pascal などの型の概念とは本質的に異なる豊かな概念である. **ML** のプログラム言語としての成功と, F の多相型に対する型理論としての成功によって, 計算機科学と数理論理学の両分野における型理論の活発な研究および交流が始まった.

その後の研究の課題の一つに, F よりも強力な型理論を作ることがあった. F のような高階の全称束縛を拡張すれば, 型理論はより強力な表現力を持つようになる. しかし, 強力に拡張し過ぎると, Girard の逆説のような矛盾が発生してしまう. この点で, 最も成功した型理論の一つが, CC である [11]. CC は, 強正規性, 意味論などの理論的側面と, その証明システムを計算機上に実現して, 数学の形式的証明やプログラム合成を計算機上で行うという実際的側面の両面で, 豊かな成果をあげている.

型理論で多く用いられる記法について説明する. $\to$ は, 右結合であるとして用い, () を省略する. 例えば,

$$A_1 \to A_2 \to A_3 \to A_4$$

によって,

$$A_1 \to (A_2 \to (A_3 \to A_4))$$

を表す. $\forall x : A$, $\Pi x : A$, $\Sigma x : A$ などの単項構成子は, $\to$, $\times$, $+$ などの二項構成子や, 関数適用よりも, 結合力が弱いとして用いる. 例えば,

$$\forall P : *.(\forall x : A.A(P)(x) \to P(x)) \to P(x)$$

によって,

$$\forall P : *.((\forall x : A.(A(P)(x) \to P(x))) \to P(x))$$

を表す.

第 2 章

単純型理論 $\lambda\!\rightarrow$

簡単な型理論の例として，単純型理論 $\lambda\!\rightarrow$ を考えよう．$\lambda\!\rightarrow$ は，通常の λ 式 [1] に型を与えることを論じる．閉じた λ 式は，それにより計算方法が与えられた関数と考えることができる．その関数の持つ型が，その λ 式の型であると考える．例えば，$\lambda x.x$ は，恒等関数を表す．したがって，定義域を A に定めると，$\lambda x.x$ は，$A \rightarrow A$ の型を持つと考える．これを，

$$\lambda x.x : A \rightarrow A$$

と表す．$\lambda\!\rightarrow$ での言明は，t を λ 式，A を型として，

$$t : A$$

の形の式である．この式は，λ 式 t が型 A の型を持つことを表す．

定義 2.1. **(単純型理論 $\lambda\!\rightarrow$)**

$\lambda\!\rightarrow$ における項は，通常の λ 式である．すなわち，項 を次のように定義する．

1. 変数は，項 である．（ここでは $x, y, \ldots$ を用いる．）
2. t_1, t_2 が 項 なら，$t_1 t_2$ は項である．
3. x が変数，t が 項 なら，$\lambda x.t$ は項 である．

型を次のように定義する．

1. 変数は型である．（ここでは $\alpha, \beta, \ldots$ を用いる．）
2. A, B が型なら，$A \rightarrow B$ は型である．

以下では，$\rightarrow$ は右結合の記号として使う．

判定 は，次の形の式である．

$$x_1 : A_1, \ldots, x_n : A_n \vdash t : A$$

ただし，$n \geq 0$，x_i は変数，A_i, A は型，t は 項 である．

$x_1 : A_1, \ldots, x_n : A_n$ の形の式を 文脈 という．

推論規則として，次の規則を持つ．

$$\Gamma \vdash x : A \qquad (\Gamma(x) = A)$$

$$\frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash fa : B}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}$$

ここで，Γ は文脈 である．$\Gamma(x) = A$ とは，Γ が，

$$x_1 : A_1, \ldots, x_n : A_n, x : A, y_1 : B_1, \ldots, y_m : B_m$$

の形をしていて，かつ，$y_i \not\equiv x \quad (1 \leq i \leq m)$ であることを表す．□

判定 $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$ は，変数 x_i の型が A_i $(1 \leq i \leq n)$ ならば，項 t の型が A であることを表す．

例えば，λ 式 $\mathbf{K} \equiv \lambda xy.x$ を考えてみよう．$\mathbf{K}$ は，$A \to B \to A$ の型を持つ．このことは，次のように推論できる．

$$\cfrac{\cfrac{x : A, y : B \vdash x : A}{x : A \vdash \lambda y.x : B \to A}}{\vdash \lambda xy.x : A \to B \to A}$$

もう一つ $\mathbf{S} \equiv \lambda xyz.xz(yz)$ の例を挙げよう．π_1 を次の証明図とする．

$$\frac{x : A \to B \to C, y : A \to B, z : A \vdash x : A \to B \to C \quad x : A \to B \to C, y : A \to B, z : A \vdash z : A}{x : A \to B \to C, y : A \to B, z : A \vdash xz : B \to C}$$

π_2 を次の証明図とする．

$$\frac{x : A \to B \to C, y : A \to B, z : A \vdash y : A \to B \quad x : A \to B \to C, y : A \to B, z : A \vdash z : A}{x : A \to B \to C, y : A \to B, z : A \vdash yz : B}$$

$\mathbf{S}$ の型についての証明は，次のようである．

$$\cfrac{\cfrac{\cfrac{\cfrac{\overset{\vdots\, \pi_1}{x : A \to B \to C, y : A \to B, z : A \vdash xz : B \to C} \quad \overset{\vdots\, \pi_2}{x : A \to B \to C, y : A \to B, z : A \vdash yz : B}}{x : A \to B \to C, y : A \to B, z : A \vdash xz(yz) : C}}{x : A \to B \to C, y : A \to B \vdash \lambda z.xz(yz) : A \to C}}{x : A \to B \to C \vdash \lambda yz.xz(yz) : (A \to B) \to A \to C}}{\vdash \lambda xyz.xz(yz) : (A \to B \to C) \to (A \to B) \to A \to C}$$

上の推論を見ると，$\mathbf{K}$ や $\mathbf{S}$ は，別の型を持つことも推論できることがわかる．例えば，上の A を $C \to D$ で，B を $E \to E$ で置き換えても，同様の推論が成り立ち，

$$\mathbf{K} : (C \to D) \to (E \to E) \to C \to D$$

が結論できる．このようにして，**K** はたくさんの型を持つ．しかし，実は，**K** の型は，すべて，$\alpha \to \beta \to \alpha$ の α, β に適当な型を代入してできた型になっている．一般に，このような型を，principal type という [24].

定義 2.2. **(principal type)**
型 A が 項 t の principal type であるとは，次が成り立つことと定める．

1. $\vdash t : A$ が成り立つ．
2. $\vdash t : B$ が成り立つなら，A 中の変数に対する代入 σ が存在して，$B \equiv A\sigma$ となる．

□

principal type は，存在すれば，自由変数の名前換えを除いて，一意的に定まることは，明らかである．
principal type について，次が成り立つ．

定理 2.3. **([24])**

1. 項 t が型を持つなら，t は principal type を持つ．
2. 項 t の principal type は計算可能である．すなわち，t を入力とし，t が principal type を持てば，その principal type を返し，t が principal type を持たないなら，持たない旨を知らせて止まるプログラムが存在する．

□

この性質により，λ 式 t が与えられると，t が principal type を持つかどうか判定し，持つならばその principal type を計算することができる．この計算は，t の長さに関して線形の計算量で計算できる．このように，与えられた式から，その式の持つ型を計算することを，型推論 (type inference) という．
λ 式 t が型 A を持つことがわかると，次のようなたくさんのよい性質がわかる．

定理 2.4. **([24])**

1. $\vdash t : A$ かつ $t \xrightarrow[\beta\eta]{*} s$ ならば，$\vdash s : A$.
2. $\vdash t : A$ ならば，t は 強正規化可能 (strongly normalizable) である．すなわち，t の $\beta\eta$- 簡約は，どの戦略によっても必ず停止する．

3. $\vdash t : A$ かつ $\vdash s : B$ なら，$t =_{\beta\eta} s$ か否かの判定は決定可能である．

□

この性質のため，型を持つ λ 式は，よい性質を持つ一部の λ 式に限られる．正規形である λ 式でさえ，必ずしも型を持つとは限らない．例えば，$\lambda x.xx$ は型を持たない．

$\lambda\!\to$ に限らず，一般に，型理論では，定理 2.3，2.4 のようなよい性質が成り立つ．また，より強力な拡張された型理論を構成する場合には，このようなよい性質が成り立つか否かが，重要な問題となる．

第 3 章

ML の型推論

プログラム言語における型は，次の利点がある．

1. 型という補助的情報をプログラムに追加することにより，人間がより理解しやすい読みやすいプログラムになる．
2. 型検査により，型が合っていないというプログラムの誤りを，コンパイルのときに発見することができる．
3. コンパイラが，型に関する情報を用いて，より効率のよいコードを生成することができる．
4. 再帰呼び出しを許す部分以外については，型を持つことが，プログラムの実行が停止することを保証する．
5. 型によりプログラムの仕様の全部または一部を与えるとき，プログラムの正当性の検証が型検査により実現される．

一方で，型付言語は，次のような欠点を持つ可能性がある．

1. プログラム中の型の情報が重要である部分だけではなく，型を持つすべての部分式にいちいち型を書くのは，プログラムを書くときにもわずらわしいし，読むときにも読みにくい．
2. Lisp の関数のように，一つの関数を様々なデータに対して用いたい場合があるが，そのようにできない．

欠点の 1. について，例えば，次の階乗を計算する関数 fact を考えてみよう．以下では， Standard ML [49] の記法を用いる

```
let val rec fact = fn x => if x = 0
        then 1 else x * fact(x - 1) in ...
```

ML では，int で整数の型を，bool で論理値の型を，A * B で直積型を，A -> B で関数型を表す．このプログラムを見れば，明らかに，

```
x:int
fact:int -> int
= :int * int -> bool
* :int * int -> int
- :int * int -> int
if - then - else - :bool * int * int -> int
```

などの型情報がわかる．(x - 1) を見ると，1:int なので，-:int * int -> int がわかり，ゆえに，x:int がわかる．これらの型情報をいちいちプログラム中に人手で書くのは，面倒であるし，プログラムが読みにくくもなる．

この解決として，型を明示的にプログラムに書くのは，必要最小限の情報で十分とし，型の与えられていない部分式は，式の性質から型を推論する方法が用いられる．この推論が，型推論 (type inference) である．ML では，型推論が行われ，原則として，変数の型宣言は不要である．

欠点の 2. について，Lisp の関数 mapcar を考えてみよう．mapcar は，リスト l と，l の要素に対する関数 f が引数として与えられると，l の各要素に f を行ってできたリストを返す．次のように使う．

```
(mapcar #'1+ '(1 2 3)) => (2 3 4)
(mapcar #'null '((1) () (2 3))) => (nil t nil)
```

ML と同様の型を考えると，mapcar の型は，それぞれ次のようになる．

```
mapcar : (int -> int) -> (int list) -> (int list)
mapcar : ((int list) -> bool) -> ((int list) list) -> (bool list)
```

ここで，型 A list は，型 A を持つ要素からなるリストの型を表す．二つの mapcar は，異なる型を持つので，型付言語では，異なった関数として扱われる可能性がある．型だけが異なるだけで文面や機能がほとんど同じである関数を，別の名前を付けて二重に書くことは，プログラムの作成，利用，保守の上で不都合である．

この欠点の解決のため，多相型 (polymorphism) が提案された [35]．型変数を含む型を 多相的 な型という．多相型 とは，多相的 な型を用いる型付けの方法である．型変数を含む型を持つ式は，その型変数に任意の型を代入した型すべてを型として持つとして扱われる．

多相型 により，関数を定義するときには，様々な型に対して一様に統一的に使える関数を定義し，関数を具体的に使うときには，各々異なる特定の型を持つ関数として

使うことができる. mapcar の型は, 型変数 'a, 'b を用いて次のように表せる.

```
mapcar : ('a -> 'b) -> ('a list) -> ('b list)
```

ML で用いられている型変数を含む型の型推論は, 計算可能である. さらに, 型変数を含む型の型推論の計算アルゴリズムは, 型変数を含まない型推論の計算アルゴリズムを少し拡張するだけで得られる. プログラム言語 ML は, 多相的 な型理論を取り入れた代表的なプログラム言語である.

第 4 章

型推論のアルゴリズム

MLの型推論のアルゴリズムを，MLの部分言語に対して示す．

次のようなMLの部分言語の構文を持つプログラム言語を考える．

定義 4.1.

式 e を次のように定める．

$$e ::= x| < \text{number} > |\texttt{true}|\texttt{false}|\lambda x.e_1|e_1 e_2|$$

$$\texttt{let } x = e_1 \texttt{ in } e_2| \texttt{ fix } x.e| \texttt{ if } e_1 \texttt{ then } e_2 \texttt{ else } e_3$$

ただし，x は変数を表す．

型 τ を次のように定める．

$$\tau ::= \alpha|\texttt{int}|\texttt{bool}|\tau_1 \to \tau_2$$

ただし，α は，型変数を表す．□

このプログラムの意味は，型付けの点を除いて，他のプログラム言語からの類推により理解してよい．

この言語の型付けを，型理論を用いて定義する．次に紹介するMLの型理論は，[12] の型理論を元にして，わかりやすさのため変形して提案された型理論である．

定義 4.2. (MLの型理論 [6])

項として，MLの式を持つ．次の型 σ を持つ．ここで，τ はMLの型を表す．

$$\sigma ::= \tau|\forall\alpha.\sigma$$

判定は，

$$x_1 : A_1, \ldots, x_n : A_n \vdash e : \tau$$

の形である．ただし，A_i $(1 \le i \le n)$ は型 σ であり，τ は ML の型である．次の推論規則を持つ．

$$\Gamma \vdash \mathrm{n}: \texttt{int} \qquad (n \quad \text{number})$$

$$\Gamma \vdash \texttt{true} : \texttt{bool}$$

$$\Gamma \vdash \texttt{false} : \texttt{bool}$$

$$\Gamma \vdash x : \tau \qquad (\Gamma(x) = \sigma, \quad \sigma \ge \tau)$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}$$

$$\frac{\Gamma, x : \tau_1 \vdash e_1 : \tau_2}{\Gamma \vdash \lambda x.e_1 : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \forall\vec{\alpha}.\tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2} (*)$$

$$\frac{\Gamma, x : \tau \vdash e : \tau}{\Gamma \vdash \texttt{fix}\ x.e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

ここで，次の記法を用いた．$\Gamma(x) = \sigma$ とは，Γ が，

$$x_1 : A_1, \ldots, x_n : A_n, x : \sigma, y_1 : B_1, \ldots, y_m : B_m$$

の形をしていて，かつ，$y_i \not\equiv x$ $(1 \le i \le m)$ であることを表す．$FV(e)$ により，e 中に現れる自由な型変数全体を表す．推論規則 (*) 中の $\vec{\alpha}$ は，$FV(\tau_1) - FV(\Gamma)$ を表す．$\sigma \ge \tau$ は，σ が

$$\forall\alpha_1 \ldots \alpha_n.\rho$$

の形をしていて，ρ には $\forall$ が現れないとき，ML の型 $\tau_1, \ldots, \tau_n$ が存在して，

$$\tau \equiv \rho[\alpha_1 := \tau_1, \ldots, \alpha_n := \tau_n]$$

であることを表す．□

上の型理論で使われる型 σ は，ML の型 τ と異なるが，ML の型を含んでいる．

ML の変数型は，let 式で実現される．let 式を使わないと，型変数を使わない型付けしか行えない．次の例を考えてみよう．計算結果の点からは，これらの 2 式は，同じ計算結果を返す．

```
let x = e1 in e2
```

$(\lambda\ x.e_2)\ e_1$

しかし，ML の型推論では，この 2 式は異なった扱いを受ける．前者の場合，型推論は，まず，e_1 の型 σ を計算する．σ 中の型変数にそれぞれ異なった型を代入した型を $\sigma_1, \ldots, \sigma_n$ とする．e_2 中に現れる各 x の型について，異なる x の出現は，各々異なった型 σ_i を持ってよい．後者の場合，e_1 の型 σ の特定の instance σ_1 が λx の x に型付けされ，e_2 中のすべての x の出現は，同じ σ_1 の型を持つ必要がある．ゆえに，前者では，e_1 が型変数を含む型を持つ一般的関数である場合，e_2 の計算において，x を型変数による柔軟な型を持つ関数として使うことができる．

次の例を考えよう．

```
let f = λx.x in f(f)
(λf.f(f))(λx.x)
```

前者は，$\alpha \to \alpha$ の型を持つ．この型推論は，次のように証明できる．

$$\dfrac{\dfrac{\mathtt{x}:\beta \vdash \mathtt{x}:\beta}{\vdash \lambda\mathtt{x}.\mathtt{x}:\beta \to \beta} \qquad \dfrac{\mathtt{f}:\forall\beta.(\beta\to\beta) \vdash \mathtt{f}:(\alpha\to\alpha)\to\alpha\to\alpha \quad \mathtt{f}:\forall\beta.(\beta\to\beta)\vdash \mathtt{f}:\alpha\to\alpha}{\mathtt{f}:\forall\beta.(\beta\to\beta)\vdash \mathtt{f(f)}:\alpha\to\alpha}}{\vdash \mathtt{let\ f} = \lambda\mathtt{x}.\mathtt{x}\ \mathtt{in\ f(f)}:\alpha\to\alpha}$$

しかし，後者は型付け不可能である．λf の f が，特定の型 $\alpha \to \alpha$ を持つことになり，f(f) が，$\alpha \to \alpha$ の型の式に，$\alpha \to \alpha$ の型の式を適用することになり，型付けが失敗する．

上の推論規則を見ると，$\Gamma \vdash e : \tau$ の証明図の最後に使われる規則は，e の形により一通りに決まってしまうことがわかる．このことより，$\Gamma \vdash e : \tau$ の証明図は，存在すれば，形は一通りであり，その形は，e の形と同形であることがわかる．このことを用いて，e と Γ が与えられると，e の型を推論するアルゴリズムが構成できる．

以下では，代入を効率よく扱うため，副作用を用いたアルゴリズムを紹介する [35]．

広域変数として E を使う．E は代入を表す．

関数 unify は，unification を行う次のような関数で，すでに与えられているとする．

- unify(σ, τ) の形で用いる．σ, τ は型である．
- unify を行うと，E が副作用により変化する．
- $E = E_1$ として，unify(σ, τ) を行うと，まず，$E_1(\sigma)$ と $E_1(\tau)$ の mgu を計算する．mgu がなければ，失敗した旨述べて，アルゴリズムの実行全体を止める．mgu U があれば，$E := UE_1$ とし，関数の実行を終わる．

型推論を行う関数 typing を次のように定める．

typing(Γ, e) の形で用いる．Γ は変数の型宣言，e は式である．広域変数 E が副作用で変化する．typing(Γ, e) は，型推論の結果の型を返すか，または，型付けに失敗して，失敗した旨述べて，アルゴリズム全体の実行を終了する．

このアルゴリズムは，次が成り立つように作られている．

命題 4.3.

$E = E_1$ のとき，typing(Γ, e) が，τ を返し，$E = E_2$ になることと，$E_1(\Gamma)$ の下で e が型付け可能で，その principal typing scheme [24] が，$E_2(\Gamma) \vdash e : E_2(\tau)$ であることは，同値である．□

定義 4.4. (typing)

$E = E_1$ のとき，typing(Γ, e) を行った場合，関数 typing の計算は，e の形による場合分けにより，次のように実行する．

- e が整数の場合．`int` を返す．
- e が，`true` または，`false` の場合．`bool` を返す．
- e が 変数 x の場合．$\Gamma(x) = \tau$ で，τ が $\forall$ を含んでいないなら，τ を返す．$\Gamma(x) = \forall\alpha_1 \ldots \alpha_n.\tau$ で，τ が $\forall$ を含んでいないなら，新しい型変数 $\beta_1, \ldots, \beta_n$ を用意して，$\tau[\alpha_1 := \beta_1, \ldots, \alpha_n := \beta_n]$ を返す．
- e が，$e_1 e_2$ の場合．

  $\rho :=$ typing$(\Gamma,\ e_1)$
  $\sigma :=$ typing$(\Gamma,\ e_2)$
  新しい型変数 β を用意する．
  unify$(\rho,\ \sigma\ \rightarrow\ \beta)$
  β を返す．

- e が，`if` e_1 `then` e_2 `else` e_3 の場合．

  $\rho :=$ typing$(\Gamma,\ e_1)$
  unify$(\rho$, bool$)$
  $\sigma :=$ typing$(\Gamma,\ e_2)$
  $\sigma' :=$ typing$(\Gamma,\ e_3)$
  unify$(\sigma,\ \sigma')$
  σ を返す．

- e が， $\lambda x.e_1$ の場合.

  新しい型変数 β を用意する.

  $\rho :=$ typing($< \Gamma,\ x\ :\ \beta >$, e_1)

  $\beta\ \rightarrow\ \rho$ を返す.

  ここで， $< \Gamma, x : \beta >$ は， $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ のとき， $x_1 : A_1, \ldots, x_n : A_n, x : \beta$ を表す.

- e が，`fix` x.e_1 の場合.

  新しい型変数 β を用意する.

  $\rho :=$ typing($< \Gamma,\ x : \beta >$, e_1)

  unify(β, ρ)

  β を返す.

- e が，`let` x=e_1 `in` e_2 の場合.

  $\rho :=$ typing(Γ, e_1)

  $\sigma :=$ typing($< \Gamma,\ x : \forall\ \vec{\alpha}.\rho >$, e_2)

  ただし， $\vec{\alpha}\ =\ FV(\rho)\ -\ FV(\Gamma)$.

  σ を返す.

□

型推論のプログラム全体は，次のように実行する.

定理 4.5. ([35])

E := 空の代入

typing(<>, e)

を実行したとき，次が成り立つ. ここで， <> は空の変数宣言を表す.

- e が型付け可能の場合.

  関数 typing(<>, e) の実行は，結果として型 τ を返す. $E = E'$ となったとすると， $E'(\tau)$ は e の principal type である.

- e が型付け不可能の場合.

  関数 typing(<>, e) の実行は，失敗した旨述べて停止する.

□

`fix` は，ML で，すべての部分帰納関数を表現するため，実際のプログラミングには必要である．しかし，`fix` のため，型を持つすべてのプログラムの実行が停止するわけではない．すなわち，ML の型理論では，`fix` があるため，型付けされた式が強正規化可能であるということは成り立たない．また，Curry-Howard の同型対応により対応する論理体系を考えると，その論理体系は矛盾している．ML の型理論では，次のようにして，すべての型 A が要素 `fix` $x.x$ を持つからである.

$$\frac{x : A \vdash x : A}{\vdash \mathtt{fix}\, x.x : A}$$

ML の型理論から `fix` を除けば，強正規化可能性が成り立ち，また，Curry-Howard の同型対応により対応する論理体系は，無矛盾である．しかし，`fix` 無しでは，すべての部分帰納関数を表現することはできなくなる.

ML の型推論で使われる多相型 は，型付プログラム言語の欠点を克服し，型付プログラム言語のよい面を活かした優れた機構である．∀ により束縛する型変数を拡張することにより強力な型理論が種々提案されている．型の機構がより強力になると，型推論がより計算可能でなくなる．型推論が計算可能であることがわかっている型変数を含む型理論の中では，ML の型理論は最も強力な型理論の一つである.

第 5 章

Martin-Löf の型理論

5.1 $\mathbf{ITT}_n$ の定義

Martin-Löf の型理論には，$\mathbf{ITT}_0, \mathbf{ITT}_1, \ldots$ がある．$\mathbf{ITT}_{n+1}$ は，$\mathbf{ITT}_n$ の拡張である．以下では，$\mathbf{ITT}_n$ $(n = 0, 1, \ldots)$ を紹介する [33].

まず，$\mathbf{ITT}_0$ を定義する．

定義 5.1. ($\mathbf{ITT}_0$ の判定)

$\mathbf{ITT}_0$ における基本的言明は，次の 4 種類の 判定 である．

A type

$a \in A$

$a = b \in A$

$A = B$

□

A type は，式 A が型であることを表す．$a \in A$ は，式 A が型であり，式 a が型 A の要素であることを表す．$a = b \in A$ は，式 A が型であり，式 a, b がともに型 A の要素であり，かつ a と b は型 A の要素として等しいことを表す．$A = B$ は，式 A, B はともに型であり，かつ，両者は等しい型であることを表す．

定義 5.2. ($\mathbf{ITT}_0$ の型)

基本型として，N_k $(k = 0, 1, \ldots)$, N を持ち，また，$(\Pi x \in A)B$, $(\Sigma x \in A)B$, $A + B$, $\mathrm{I}(a, b, A)$ の構成型を持つ．□

N_k は有限集合に相当する型である．N は自然数全体に相当する型である．$(\Pi x \in A)B(x)$ は，$a \in A$ が与えられたとき，$f(a)$ の値が a によって異なる型 $B(a)$ に属

する一般的関数 f の型である．$(\Sigma x \in A)B(x)$ は，$a \in A$ であり，b は a によって異なる型 $B(a)$ に属する一般的対 $< a, b >$ の型である．$A + B$ は，直和 の型を表す．$\mathrm{I}(A, a, b)$ は，判定 $a = b \in A$ を表す型である．

各型について，各々 形成規則 (formation rule), 導入規則 (introduction rule), 除去規則 (elimination rule), 等号規則 (equality rule) の推論規則がある．形成規則 は，式がその型であるための必要十分条件を述べる．導入規則 は，その型の canonical element が何であるかを述べる．canonical element とは，λ 計算での最左簡約に関する正規形に対応する要素で，値を表す．除去規則 は，その型の要素に関する帰納法について述べる．等号規則 は，正規化またはカット除去について述べる．

定義 5.3. **(ITT_0 の推論規則)**

$$\frac{a \in A}{a = a \in A} \qquad \frac{A\ \mathrm{type}}{A = A}$$

$$\frac{a = b \in A}{b = a \in A} \qquad \frac{A = B}{B = A}$$

$$\frac{a = b \in A \quad b = c \in A}{a = c \in A} \qquad \frac{A = B \quad B = C}{A = C}$$

$$\frac{a \in A \quad \begin{matrix}[x \in A]\\ \vdots \\ B(x)\ \mathrm{type}\end{matrix}}{B(a)\ \mathrm{type}} \qquad \frac{a = c \in A \quad \begin{matrix}[x \in A]\\ \vdots \\ B(x)\ \mathrm{type}\end{matrix}}{B(a) = B(c)}$$

$$\frac{a \in A \quad A = B}{a \in B} \qquad \frac{a \in A \quad \begin{matrix}[x \in A]\\ \vdots \\ B(x) = D(x)\end{matrix}}{B(a) = D(a)}$$

$$\frac{a \in A \quad \begin{matrix}[x \in A]\\ \vdots \\ b(x) \in B(x)\end{matrix}}{b(a) \in B(a)} \qquad \frac{a = c \in A \quad \begin{matrix}[x \in A]\\ \vdots \\ b(x) \in B(x)\end{matrix}}{b(a) = b(c) \in B(a)}$$

$$\frac{a \in A \quad \begin{matrix}[x \in A]\\ \vdots \\ b(x) = d(x) \in B(x)\end{matrix}}{b(a) = d(a) \in B(a)}$$

Π に関する推論規則は次の通りである．

$$(\Pi-\text{form})\qquad \frac{A \text{ type} \quad \begin{matrix}[x\in A]\\ \vdots \\ B(x) \text{ type}\end{matrix}}{(\Pi x\in A)B(x) \text{ type}}$$

$$\frac{A=C \quad \begin{matrix}[x\in A]\\ \vdots \\ B(x)=D(x)\end{matrix}}{(\Pi x\in A)B(x)=(\Pi x\in C)D(x)}$$

$$(\Pi-\text{intro})\qquad \frac{A \text{ type} \quad \begin{matrix}[x\in A]\\ \vdots \\ b(x)\in B(x)\end{matrix}}{\lambda x.b(x)\in(\Pi x\in A)B(x)}$$

$$\frac{A \text{ type} \quad \begin{matrix}[x\in A]\\ \vdots \\ b(x)=d(x)\in B(x)\end{matrix}}{\lambda x.b(x)=\lambda x.d(x)\in(\Pi x\in A)B(x)}$$

$$(\Pi-\text{elim})\qquad \frac{c\in(\Pi x\in A)B(x) \quad a\in A}{\text{Ap }(c,a)\in B(a)}$$

$$\frac{c=d\in(\Pi x\in A)B(x) \quad a=b\in A}{\text{Ap }(c,a)=\text{Ap }(d,b)\in B(a)}$$

$$(\Pi-\text{equal})\qquad \frac{a\in A \quad \begin{matrix}[x\in A]\\ \vdots \\ b(x)\in B(x)\end{matrix}}{\text{Ap }(\lambda x.b(x),a)=b(a)\in B(a)}$$

$$\frac{c\in(\Pi x\in A)B(x)}{c=\lambda x.\text{ Ap }(c,x)\in(\Pi x\in A)B(x)}$$

Π に関する各規則の二つ目にある等号に関する規則に対応する規則は，上と同様のものであって明らかなので，以下の推論規則では省略する．その他の推論規則は，次の通りである．

$$(\Sigma-\text{form})\qquad \frac{A \text{ type} \quad \begin{matrix}[x\in A]\\ \vdots \\ B(x) \text{ type}\end{matrix}}{(\Sigma x\in A)B(x) \text{ type}}$$

$$(\Sigma-\text{intro})\qquad \frac{\begin{array}{c}[x\in A]\\ \vdots\\ B(x)\ \text{type}\end{array}\quad a\in A\quad b\in B(a)}{(a,b)\in(\Sigma x\in A)B(x)}$$

$$(\Sigma-\text{elim})\qquad \frac{c\in(\Sigma x\in A)B(x)\quad \begin{array}{c}[x\in A]\quad[y\in B(x)]\\ \vdots\\ d(x,y)\in C((x,y))\end{array}\quad \begin{array}{c}[z\in(\Sigma x\in A)B(x)]\\ \vdots\\ C(z)\ \text{type}\end{array}}{\mathrm{E}(c,(x,y)d(x,y))\in C(c)}$$

$$(\Sigma-\text{equal})\qquad \frac{\begin{array}{c}[x\in A]\\ \vdots\\ B(x)\ \text{type}\end{array}\quad a\in A\quad b\in B(a)\quad \begin{array}{c}[x\in A]\quad[y\in B(x)]\\ \vdots\\ d(x,y)\in C((x,y))\end{array}}{\mathrm{E}((a,b),(x,y)d(x,y))=d(a,b)\in C((a,b))}$$

$$(+-\text{form})\qquad \frac{A\ \text{type}\quad B\ \text{type}}{A+B\ \text{type}}$$

$$(+-\text{intro})\qquad \frac{a\in A}{\mathrm{i}(a)\in A+B}$$

$$\frac{b\in B}{\mathrm{j}(b)\in A+B}$$

$$(+-\text{elim})\qquad \frac{c\in A+B\quad \begin{array}{c}[x\in A]\\ \vdots\\ d(x)\in C(\mathrm{i}(x))\end{array}\quad \begin{array}{c}[y\in B]\\ \vdots\\ e(y)\in C(\mathrm{j}(y))\end{array}\quad \begin{array}{c}[z\in A+B]\\ \vdots\\ C(z)\ \text{type}\end{array}}{\mathrm{D}(c,(x)d(x),(y)e(y))\in C(c)}$$

$$(+-\text{equal})\qquad \frac{a\in A\quad B\ \text{type}\quad \begin{array}{c}[x\in A]\\ \vdots\\ d(x)\in C(\mathrm{i}(x))\end{array}\quad \begin{array}{c}[y\in B]\\ \vdots\\ e(y)\in C(\mathrm{j}(y))\end{array}}{\mathrm{D}(\mathrm{i}(a),(x)d(x),(y)e(y))=d(a)\in C(\mathrm{i}(a))}$$

$$\frac{A\ \text{type}\quad b\in B\quad \begin{array}{c}[x\in A]\\ \vdots\\ d(x)\in C(\mathrm{i}(x))\end{array}\quad \begin{array}{c}[y\in B]\\ \vdots\\ e(y)\in C(\mathrm{j}(y))\end{array}}{\mathrm{D}(\mathrm{j}(b),(x)d(x),(y)e(y))=e(b)\in C(\mathrm{j}(b))}$$

$$(\mathrm{I}-\text{form})\qquad \frac{A\ \text{type}\quad a\in A\quad b\in A}{\mathrm{I}(A,a,b)\ \text{type}}$$

$$(\mathrm{I}-\text{intro})\qquad \frac{a=b\in A}{\mathrm{r}\in\mathrm{I}(A,a,b)}$$

$$(\mathrm{I}-\text{elim})\qquad \frac{c\in\mathrm{I}(A,a,b)}{a=b\in A}$$

(I − equal)
$$\frac{c \in \mathrm{I}(A,a,b)}{c = \mathbf{r} \in \mathrm{I}(A,a,b)}$$

(N_k − form) N_k type

(N_k − intro) $\mathrm{m}_k \in \mathrm{N}_k \quad (m = 0,1,\ldots,k-1)$

(N_k − elim)
$$\frac{c \in \mathrm{N}_k \quad c_m \in C(\mathrm{m}_k) \quad (m = 0,1,\ldots,k-1) \quad \begin{matrix}[z \in \mathrm{N}_k]\\ \vdots \\ C(z)\ \text{type}\end{matrix}}{\mathrm{R}_k(c, c_0, \ldots, c_{k-1}) \in C(c)}$$

(N_k − equal)
$$\frac{c_m \in C(\mathrm{m}_k) \quad (m = 0,1,\ldots,k-1)}{\mathrm{R}_k(\mathrm{m}_k, c_0, \ldots, c_{k-1}) = c_m \in C(\mathrm{m}_k)}$$

(N − form) N type

(N − intro)
$$0 \in \mathrm{N} \qquad \frac{t \in \mathrm{N}}{\mathrm{succ}(t) \in \mathrm{N}}$$

(N − elim)
$$\frac{n \in \mathrm{N} \quad d \in C(0) \quad \begin{matrix}[x \in \mathrm{N}] \quad [y \in C(x)]\\ \vdots \\ e(x,y) \in C(\mathrm{succ}(x))\end{matrix} \quad \begin{matrix}[z \in \mathrm{N}]\\ \vdots \\ C(z)\ \text{type}\end{matrix}}{\mathrm{R}\ (n, d, (x,y)e(x,y)) \in C(n)}$$

(N − equal)
$$\frac{A\ \text{type} \quad d \in C(0) \quad \begin{matrix}[x \in A] \quad [y \in C(x)]\\ \vdots \\ e(x,y) \in C(\mathrm{succ}(x))\end{matrix}}{\mathrm{R}\ (0, d, (x,y)e(x,y)) = d \in C(0)}$$

$$\frac{A\ \text{type} \quad a \in \mathrm{N} \quad d \in C(0) \quad \begin{matrix}[x \in A] \quad [y \in C(x)]\\ \vdots \\ e(x,y) \in C(\mathrm{succ}(x))\end{matrix}}{\mathrm{R}\ (\mathrm{succ}(a), d, (x,y)e(x,y)) = e(a, \mathrm{R}\ (a, d, (x,y)e(x,y))) \in C(\mathrm{succ}(a))}$$

□

定義 5.4. ($\mathbf{ITT}_n$ $\quad (n = 1,2,\ldots)$)

$\mathbf{ITT}_1$ は，次の規則を追加して $\mathbf{ITT}_0$ を拡張した型理論である．

(U_0 − form) U_0 type

(U_0 − intro)
$$\frac{A \in \mathrm{U}_0 \quad \begin{matrix}[x \in A]\\ \vdots \\ B(x) \in \mathrm{U}_0\end{matrix}}{(\Pi x \in A)B(x) \in \mathrm{U}_0}$$

$$\frac{A \in \mathrm{U}_0 \quad \begin{array}{c}[x \in A] \\ \vdots \\ B(x) \in \mathrm{U}_0\end{array}}{(\Sigma x \in A)B(x) \in \mathrm{U}_0}$$

$$\frac{A \in \mathrm{U}_0 \quad B \in \mathrm{U}_0}{A + B \in \mathrm{U}_0}$$

$$\frac{A \in \mathrm{U}_0 \quad a \in A \quad b \in A}{\mathrm{I}(A, a, b) \in \mathrm{U}_0}$$

$$\mathrm{N}_k \in \mathrm{U}_0 \qquad (k = 0, 1, \ldots)$$

$$\mathrm{N} \in \mathrm{U}_0$$

(U_0-elim), (U_0-equal) については，(U_0-intro) から自然に作られる規則であり，ここでは省略する.

$n = 1, 2, \ldots$ について，$\mathbf{ITT}_{n+1}$ は，次の規則を追加して $\mathbf{ITT}_n$ を拡張した型理論である.

$(\mathrm{U}_n - \mathrm{form}) \qquad \mathrm{U}_n$ type

$(\mathrm{U}_n - \mathrm{intro}) \qquad \mathrm{U}_{n-1} \in \mathrm{U}_n$

$$\frac{A \in \mathrm{U}_{n-1}}{A \in \mathrm{U}_n}$$

$$\frac{A \in \mathrm{U}_n \quad \begin{array}{c}[x \in A] \\ \vdots \\ B(x) \in \mathrm{U}_n\end{array}}{(\Pi x \in A)B(x) \in \mathrm{U}_n}$$

$$\ldots$$

(U_n-intro) の他の規則は (U_0-intro) と同様である. (U_n-elim), (U_n-equal) については，(U_n-intro) から自然に作られる規則であり，ここでは省略する. □

U_n は，N, N_k, U_0, …, U_{n-1} から，Π, Σ, +, I によって構成される型全体からなる型である.

5.2 $\mathbf{ITT}_n$ の目的

$\mathbf{ITT}_n$ で Martin-Löf が目的としたことは次のことであった.

1. 論理体系を，数学的対象として扱うのではなく，数学に基礎を与える論理体系を構成する．
2. 論理体系の意味は，モデルによる解釈という数学的手段により与えられるのではなく，その論理体系自体の意味を明晰に説明することによって与えられる．

1. のため，2. は必然である．論理体系の意味をモデルにより解釈することにより与えると，モデルに関して用いた数学の意味を，$\mathbf{ITT}_n$ によって与えることができなくなり，1. が実現できない．

2. のため，$\mathbf{ITT}_n$ は，できる限り，自然で明解な概念と推論規則だけを使っている．それらの概念と規則は，すべて，単純明快なことがらから始めて下のレベルから順々に積み上げて構成される．$\mathbf{ITT}_n$ は predicative な型理論とよばれる．$\mathbf{ITT}_n$ は，型全体の型というような predicative でない超越的な概念を一切含んでいない．

型理論 $\mathbf{ITT}_n$ を構成した目的の一つは，型理論 $\mathbf{ITT}_n$ を用いて通常の論理体系を解釈することであった．解釈は，論理式を型で解釈し，論理式が真であることを論理式に対応する型が要素を持つことによって解釈する．論理式を構成する論理記号は，型を構成する型構成子により解釈される．証明できる論理式 A は，論理体系中で証明図 P を持つ．A が型 A' で解釈されるとする．P が与えられると，$a \in A'$ となるような式 a を P から構成することができる．Martin-Löf は，この型理論を使って一階述語論理の論理記号に意味を与えた．この意味論は，discharge および自由変数だけを使って一階述語論理の論理記号に直観主義的な意味を与える点で，意義がある．

論理記号の解釈の例として $\rightarrow$ の場合を説明しよう．一階述語論理では，$\rightarrow$ に関して次のことが成り立つ．ここで，A form は式 A が論理式であることを表し，A true は式 A が真である論理式であることを表す．

$$\frac{A\ \text{form} \quad B\ \text{form}}{A \rightarrow B\ \text{form}} \qquad \frac{A\ \text{form} \quad B\ \text{form} \quad \begin{array}{c}[A\ \text{true}]\\ \vdots \\ B\ \text{true}\end{array}}{A \rightarrow B\ \text{true}}$$

$$\frac{A \rightarrow B\ \text{true} \quad A\ \text{true}}{B\ \text{true}}$$

これらの推論規則の意味は，次のようである．A, B がともに論理式ならば，$A \rightarrow B$ も論理式である．A, B がともに論理式であり，A が真な論理式であることを仮定すると B が真な論理式であることがわかるなら，$A \rightarrow B$ は真な論理式である．$A \rightarrow B$, A が真な論理式であれば，B も真な論理式である．A form に A type を対応させ，A true に A inhabited を対応させ，論理式 A, B に型 A', B' を対応させるときに論

理式 $A \to B$ に型 $(\Pi x \in A')B'$ を対応させて，上の推論規則を解釈すると，次のようになる．ただし，A inhabited は，ここで一時的に用いる新しい判定で，ある a が存在して $a \in A$ であることを表すようにこれに関する $\mathbf{ITT}_n$ の推論規則が与えられているとする．

$$\frac{A \text{ type} \quad B \text{ type}}{(\Pi x \in A)B \text{ type}}$$

$$\frac{A \text{ type} \quad B \text{ type} \quad \begin{array}{c}[A \text{ inhabited}]\\ \vdots \\ B \text{ inhabited}\end{array}}{(\Pi x \in A)B \text{ inhabited}}$$

$$\frac{(\Pi x \in A)B \text{ inhabited} \quad A \text{ inhabited}}{B \text{ inhabited}}$$

これの推論規則は，$\mathbf{ITT}_n$ の中で導出できる．

同様にして，論理記号 &, $\vee$, $\forall$, $\exists$, $=$ は，型構成子 Σ, $+$, Π, Σ, I により解釈できる．

5.3 $\mathbf{ITT}_n$ での証明例

$\mathbf{ITT}_0$ での推論の例として Σ の射影について考えてみよう．

命題 5.5.

射影を次のように定める．

$$\mathrm{p}(z) \equiv \mathrm{E}(z, (x, y)x)$$

$$\mathrm{q}(z) \equiv \mathrm{E}(z, (x, y)y)$$

次が成り立つ．

$$\frac{c \in (\Sigma x \in A)B(x)}{\mathrm{p}(c) \in A}$$

$$\frac{\begin{array}{c}[x \in A]\\ \vdots \\ B(x) \text{ type}\end{array} \quad a \in A \quad b \in B(a)}{\mathrm{p}((a, b)) = a \in A}$$

$$\frac{c \in (\Sigma x \in A)B(x)}{\mathrm{q}(c) \in B(\mathrm{p}(c))}\ (A)$$

$$\dfrac{\begin{array}{c}[x \in A] \\ \vdots \\ B(x)\ \mathrm{type}\end{array} \quad a \in A \quad b \in B(a)}{\mathrm{q}((a,b)) = b \in B(a)}$$

□

(A) 以外の規則は，$(\Sigma-\mathrm{elim})$, $(\Sigma-\mathrm{equal})$ の特別な場合であり，明らかである．(A) が導出できることを次に示そう．

$$\dfrac{c \in (\Sigma x \in A)B(x) \qquad \dfrac{\begin{array}{c}(3)\\ y \in B(x)\end{array} \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}(1)\\ x \in A \\ \vdots \\ B(x)\ \mathrm{type}\end{array} \quad \begin{array}{c}(3)\\ x \in A\end{array} \quad \begin{array}{c}(3)\\ y \in B(x)\end{array}}{\mathrm{p}((x,y)) = x \in A}\,(1)}{x = \mathrm{p}((x,y)) \in A} \qquad \begin{array}{c}(2)\\ x \in A \\ \vdots \\ B(x)\ \mathrm{type}\end{array}}{B(x) = B(\mathrm{p}((x,y)))}\,(2)}{y \in B(\mathrm{p}((x,y)))}}{\mathrm{E}(c,(x,y)y) \in B(\mathrm{p}(c))}\,(3)$$

ITT$_0$ での推論の次の例として，選択公理 AC について考えてみよう．ここでは，AC として次の形の選択公理を扱う．

命題 5.6.

$$\begin{aligned}&(\forall x \in A)(\exists y \in B(x))C(x,y) \rightarrow \\ &\qquad (\exists f \in (\Pi x \in A)B(x))(\forall x \in A)C(x, \mathrm{Ap}\,(f,x))\ \mathrm{true}\end{aligned}$$

を **ITT**$_0$ で解釈した判定は，**ITT**$_0$ で証明できる．□

この式の意味は，型 A の任意の要素 x が与えられたとき，x に依存するかもしれない型 $B(x)$ について型 $B(x)$ の要素 y が存在して命題 $C(x,y)$ が成り立つなら，型として関数型 $(\Pi x \in A)B(x)$ を持つ関数 f が存在して，型 A の任意の要素 x が与えられると命題 $C(x,y)$ が成立するような型 $B(x)$ の要素 y が，x から $\mathrm{Ap}\,(f,x)$ により得られることを表している．

AC true を **ITT**$_0$ で解釈すると，次のようになる．

$$\begin{aligned}&(\Pi z \in (\Pi x \in A)(\Sigma y \in B(x))C(x,y)) \\ &\qquad (\Sigma f \in (\Pi x \in A)B(x))(\Pi x \in A)C(x, \mathrm{Ap}\,(f,x))\ \mathrm{inhabited}\end{aligned}$$

この 判定 は， $\mathbf{ITT}_0$ の中で次のように証明できる．

$$
\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\dfrac{\overset{(2)}{z \in (\Pi x \in A)(\Sigma y \in B(x))C(x,y)} \quad \overset{(1)}{x \in A}}{\mathrm{Ap}\,(z,x) \in (\Sigma y \in B(x))C(x,y)} \\ \vdots \\ \mathrm{q}(\mathrm{Ap}\,(z,x)) \in C(x,\mathrm{p}(\mathrm{Ap}\,(z,x)))\end{array} \quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\dfrac{\overset{(2)}{z \in (\Pi x \in A)(\Sigma y \in B(x))C(x,y)} \quad \overset{(1)}{x \in A}}{\mathrm{Ap}\,(z,x) \in (\Sigma y \in B(x))C(x,y)} \\ \vdots \\ \mathrm{p}(\mathrm{Ap}\,(z,x)) \in B(x)\end{array}}{\mathrm{Ap}\,(\lambda x.\mathrm{p}(\mathrm{Ap}\,(z,x)),x) = \mathrm{p}(\mathrm{Ap}\,(z,x)) \in B(x)}}{C(x,\mathrm{Ap}\,(\lambda x.\mathrm{p}(\mathrm{Ap}\,(z,x)),x)) = C(x,\mathrm{p}(\mathrm{Ap}\,(z,x)))}}{C(x,\mathrm{p}(\mathrm{Ap}\,(z,x))) = C(x,\mathrm{Ap}\,(\lambda x.\mathrm{p}(\mathrm{Ap}\,(z,x)),x))}}{\mathrm{q}(\mathrm{Ap}\,(z,x)) \in C(x,\mathrm{Ap}\,(\lambda x.\mathrm{p}(\mathrm{Ap}\,(z,x)),x))}\,(1)}{\lambda x.\mathrm{q}(\mathrm{Ap}\,(z,x)) \in (\Pi x \in A)C(x,\mathrm{Ap}\,(\lambda x.\mathrm{p}(\mathrm{Ap}\,(z,x)),x))}}{<\lambda x.\mathrm{p}(\mathrm{Ap}\,(z,x)),\lambda x.\mathrm{q}(\mathrm{Ap}\,(z,x))> \in (\Sigma f \in (\Pi x \in A)B(x))(\Pi x \in A)C(x,\mathrm{Ap}\,(f,x))}}{\lambda z.<\lambda x.\mathrm{p}(\mathrm{Ap}\,(z,x)),\lambda x.\mathrm{q}(\mathrm{Ap}\,(z,x))> \in (\Pi z \in (\Pi x \in A)(\Sigma y \in B(x))C(x,y))(\Sigma f \in (\Pi x \in A)B(x))(\Pi x \in A)C(x,\mathrm{Ap}\,(f,x))}\,(2)
$$

ここでは，簡単のため，各規則の A type の形の仮定については省略した．

5.4 論理記号の拡張

この型理論により与えられる論理記号の意味を元にすると，一階述語論理の意味を拡張した新しい論理体系が構成できる．この論理体系では，例えば $\to$ の論理記号の意味が拡張される． $\to$ に対して，次のような新しい推論規則が用いられる．

$$
\dfrac{A\ \mathrm{form} \quad \begin{array}{c}[A\ \mathrm{true}] \\ \vdots \\ B\ \mathrm{form}\end{array}}{A \to B\ \mathrm{form}}
$$

この規則は， A が論理式であり，かつ， A が真であると仮定すれば B が論理式であることが導けるなら， $A \to B$ が論理式であることを示している．この規則は，上に示した通常の一階述語論理の推論規則の拡張になっている． A が真でない論理式ならば， B が論理式でなくとも， $A \to B$ は論理式である．論理式を真偽などの意味を持つ式であるように定義するならば，この拡張は自然なものである． A が論理式であり， B が論理式以外の式の場合， A が真でないことがわかった時点で B の意味を問うことなしに $A \to B$ が真であると考えることができる．この考え方にしたがうと，この場合にも $A \to B$ を論理式であると定義することができる．

論理記号の意味の拡張は， & についても可能である．通常の一階述語論理では，

$$
\dfrac{A\ \mathrm{form} \quad B\ \mathrm{form}}{A\ \&\ B\ \mathrm{form}}
$$

である．この規則を $\mathbf{ITT}_0$ によって解釈すると，導出可能な推論規則になる．この規則を次の規則に拡張することができる．

$$
\dfrac{A\ \mathrm{form} \quad \begin{array}{c}[A\ \mathrm{true}] \\ \vdots \\ B\ \mathrm{form}\end{array}}{A\ \&\ B\ \mathrm{form}}
$$

この規則は，A が論理式であり，かつ，A が真である論理式であると仮定すると B が論理式であることが導けるとき，$A\,\&\,B$ は論理式であることを表している．この規則によると，A が真でない論理式で，B が論理式でない式の場合にも，$A\,\&\,B$ は論理式である．これは，A が真でない論理式ならば，$A\,\&\,B$ にも真でないという意味を与えるのが自然であり，ゆえに，この場合にも式 $A\&B$ を論理式と考えるのが自然であるという考え方に合う．

$\rightarrow$ の場合と同様に，論理体系におけるこれらの新しい推論規則を型理論で解釈した結果の推論規則は，また $\mathbf{ITT}_0$ で導出できる．

5.5 プログラミングシステムへの応用

一般に，型理論は，プログラムの合成，検証に用いることができる．型理論は，Curry-Howard の同型対応により，論理体系に対応する [25]．プログラムの仕様を記述するための十分な表現力を持つ論理体系に対応する型理論は，型を用いてプログラムの仕様を表現することができる．プログラムの仕様を表す型を A とし，プログラムを t とすると，

$$t : A$$

は，プログラム t が仕様 A を満たすことを表す．ゆえに，$t : A$ を型理論において証明することは，プログラム t が仕様 A を満たすという検証の証明になっている．この方法により，プログラムの検証が型理論における型検査により実現される．

$\mathbf{ITT}_n$ は，Curry-Howard の同型対応により，直観主義論理に基づき算術を含んでいるある論理体系に対応している．構成的数学の多くは，この論理体系の中で形式化できる．この方法を用いて，構成的数学を $\mathbf{ITT}_n$ によって形式化することができる．特に，プログラムの仕様の記述に必要なデータや命題を形式化することができ，プログラムの仕様を表現することができる．

プログラムの仕様を入出力関係を表す論理式 $P(x,y)$ で表したとすると，入力データの型を A, 出力データの型を B, $P(x,y)$ に対応する型を $C(x,y)$ として，このプログラムの仕様は次の型で表される．

$$(\Pi x \in A)(\Sigma y \in B)C(x,y)$$

この型の要素 e が得られると，

$$f \equiv \lambda x.\mathrm{p}(\mathrm{Ap}\ (e,x)), \qquad r \equiv \lambda x.\mathrm{q}(\mathrm{Ap}\ (e,x))$$

と定めることにより，プログラム f とその検証の証明 r が得られる．型 A のデータ a が入力として与えられると， $\mathrm{Ap}\,(f,a)$ は型 B のデータであり，さらに

$$\mathrm{Ap}\,(r,a) \in C(a, \mathrm{Ap}\,(f,a))$$

が成り立つ．プログラム f の検証である $C(a, \mathrm{Ap}\,(f,a))$ に対応する論理式 $P(a, \mathrm{Ap}\,(f,a))$ の証明が $\mathrm{Ap}\,(r,a)$ から得られる．

Martin-Löf の型理論に基づいたプログラミングシステムの代表としては， Nuprl がある [7]． Nuprl は， $\mathbf{ITT}_n$ $(n = 0, 1, \ldots)$ に基づいて設計された自動証明システムであり，また，プログラム合成システムである．推論の型の部分だけが表示され，型を論理式とみなした場合の論理式をシステムの支援により会話的に証明する．型の要素の部分はシステム内部に蓄えられる．証明が終ると，証明に対応する要素が作られる．この要素は証明できた論理式に対応する型に属する．プログラムの仕様を入力条件と入出力関係を表す論理式で表現し，この論理式を Nuprl によって証明すると，その証明から仕様に合うプログラムを抽出することができる． Nuprl の場合，プログラム言語 **ML** で書かれたプログラムが作り出される．

Nuprl を用いた計算の例で面白いものとしては， Girard の 逆説 に関係した Y オペレータの計算がある． Girard は，すべての型の集まり τ もまた一つの型であるとして構成した型理論 $\lambda^{\tau\tau}$ が矛盾していることを証明した [15, 26, 34] が，この証明を用いて Y オペレータが $\lambda^{\tau\tau}$ で型付けできるか否かが話題となった． Nuprl 上で，この 逆説 の証明を行い，この証明からプログラムを抽出して調べることにより， Girard の 逆説 の証明から構成されるプログラムが， Y オペレータに似ているが Y オペレータそのものではないことが報告された [26]．この証明を Nuprl 上で形式化して行うことは 2 日間でなされた．できた証明から抽出されたプログラムの長さは，空白無しで紙に印刷しても， 4 0 ページの長さに及ぶものであった．このようなプログラムを証明から抽出し調べることは，計算機の支援無しに人手だけでは不可能であろう．

第 6 章

F

Girard の構成した型理論 F は，型変数による束縛を許す初めての型理論であった [14, 15, 18]. F は，本来は高階論理のカット除去定理の証明のために作られたが，計算機科学の分野でも多相型の形式化のために非常に役立つ道具となった. F は計算機科学の現在の活発な型理論の研究の出発点になった. F は数理論理学の分野で Girard により最初に構成されたが，その後まもなく計算機科学の分野でも同様の体系である多相的型付ラムダ計算 (polymorphic typed λ-calculus) が Reynolds により独立に構成された [39]. 計算機科学の分野では，F を 2 階型付ラムダ計算 (second order typed λ-calculus) または多相的型付ラムダ計算ともよぶ.

F は，単純型理論 $\lambda\!\rightarrow$ に型変数 α，項 t の型変数 α による λ 抽象 $\Lambda\alpha.t$ と型 A の型変数 α による全称束縛 $\forall\alpha.A$ を追加して拡張した型理論である. F は CC の部分となる型理論である.

F の定義を，元々の定義 [14] とは大分異なった形ではあるが，筆者がわかりやすいと思う形で紹介する. 型変数による束縛の機能を持つ型理論について，その基本的性質や，それが実際にどのように有用に使えるのかについては，CC の所でまとめて説明する.

定義 6.1. (F の式)

項，型，文脈を次のように定義する.

$$\begin{array}{rrcl} \text{項} & t & ::= & x|\lambda x:A.t|\Lambda\alpha.t|t_1t_2|tA \\ \text{型} & A & ::= & \alpha|A_1\rightarrow A_2|\forall\alpha.A \\ \text{文脈} & \Gamma & ::= & <>\,|<\Gamma,x:A> \end{array}$$

ここで，x は変数，α は型変数を表す. □

文脈 は，変数の型宣言であり，$x_1 : A_1, \ldots, x_n : A_n$ の形をしている．ここで，x_i は相異なる変数であり，A_i は型である．

定義 6.2. (F の判定)

判定を次のように定義する．

$$\Gamma \vdash t : A$$

ここで，Γ は 文脈，t は 項，A は型である．□

$\Gamma \vdash t : A$ は，Γ の変数宣言の下で項 t が型 A に属することを表している．

定義 6.3. (F の推論規則)

推論規則は，次のようである．

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x : A.t) : A \to B}$$

$$\frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash fa : B}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \Lambda\alpha.t : \forall\alpha.A}\ (\mathrm{A})$$

$$\frac{\Gamma \vdash t : \forall\alpha.A}{\Gamma \vdash tB : A[\alpha := B]}\ (\mathrm{B})$$

ここで，Γ は文脈，x は変数，A,B は型，t は項を表す．

ただし，規則 (A) では，型変数 α が Γ 中に自由に現れてはいけない．また，規則 (B) では，$A[\alpha := B]$ は，型 A 中にある型変数 α のすべての自由な出現を型 B で置き換えた型を表す．□

多相型は，3，4 章で説明したように，自由な型変数を含む型である．項 t が型 σ を持ち，型 σ が自由な型変数 α を持つとき，σ 中の α に他の型 τ を代入した型 $\sigma[\alpha := \tau]$ も項 t の型である．このことを表すため，4 章で行ったように，項 t が型 $\forall\alpha.\sigma$ を持つとして形式化するのが自然である．ML の型理論では $\forall\alpha$ を型の最も外側にだけ付けることができた，と考えることができる．これを一般化して，型 $\forall\alpha.\alpha \to \forall\beta.\beta$ のような型も使えるように $\forall\alpha$ を型の構成に自由に使えるように型を拡張することが考えられる．例えば，型 σ を次のように定義することを考えよう．

$$\sigma ::= \mathtt{int} | \mathtt{bool} | \alpha | \sigma_1 \to \sigma_2 | \forall\alpha.\sigma$$

この型は ML の多相型を含んでいる．このような型自体も多相型とよぶ場合がある．F はこのような多相型に対応する型理論である．

F では，λ 抽象された項は，A を型，t を項として，$\lambda x : A.t$ の形をしている．λ 抽象により項が作られるときには，抽象される変数 x の型 A が明示的に項中に示されている．このような項に対する型付けを明示的型付けという．F や CC は明示的型付けを行う型理論である．一方で，推論規則

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}$$

のように，抽象する変数 x の型を示さない項を用いる型付けもある．このような型付けを黙示的型付けという．ML の型理論や $\mathbf{ITT}_n$ では黙示的型付けが用いられている

ML の型理論は F に比べると複雑な型理論になっている．これは，F が自然で美しい型理論を目的としているのに対して，ML の型理論は現実のプログラム言語 ML に一致する型理論を目的としたためである．F には不自然な制限が無いため，ML の型理論より F の方が考えやすい場合もある．ML の型理論は，F から，型中の $\forall\alpha$ が型の最も外側にだけ付くと制限し，かつ，明示的型付けを黙示的型付けに変更することにより得られる．F は，ML の型理論に非常に近い，理論的に考えやすい型理論である．F を出発点としてこれを変形して得られた変種である型理論を研究することにより，多相型に関する型推論，強正規化可能性や意味論が明らかにされてきた [11, 30].

ML の型推論は決定可能であったが，F を黙示的型付けに変更した型理論の型推論が決定可能であるか否かは未解決問題である．

第 7 章

Calculus of Constructions

7.1 定義と基本的性質

Coquand の Calculus of Constructions (CC とよぶ) を紹介する [9, 11]. CC の表現は様々なバージョンがあるが，ここでは筆者がわかりやすいと思う形で紹介する.

CC の特徴の一つは型を要素として持つものである種 (kind) があることである. いわば，種 は型の型である. 通常の型全体を表す 種 として $*$ がある. また，型の要素の上を動く変数の他に，種 の要素である型の上を動く変数がある. 後者は型変数の役割を果たす. これらの 2 種類の変数が，両方とも Π, λ によって束縛できる. このことにより CC は他の型理論に比べて強力な型理論である. CC は，Curry-Howard の同型対応により，高階直観主義論理に対応する.

定義 7.1. (CC の式)

CC は，次の式を使う.

$$
\begin{array}{rrcl}
\text{項} & M & ::= & x|\lambda x:A.M|\lambda x:K.M|M_1M_2|MA \\
\text{型} & A & ::= & x|\Pi x:A_1.A_2|\Pi x:K.A|\lambda x:A_1.A_2|\lambda x:K.A|AM|A_1A_2 \\
\text{種} & K & ::= & *|\Pi x:A.K|\Pi x:K_1.K_2 \\
\text{文脈} & \Gamma & ::= & <>|<\Gamma,x:A>|<\Gamma,x:K>
\end{array}
$$

A,B を型または種，x を変数，x が B 中に自由に現れないとき，$\Pi x:A.B$ を $A{\to}B$ と表す. □

CC では，項，型，種，文脈 の四つの概念を用いる. 項，型は他の型理論と同様である. 種 は，型を要素として持つもので，いわば，型の型である. 型 A が 種 K に

属すことを，項 が型に属すことを表すときと同じ記号を用いて，$A : K$ で表す．文脈は，変数の型宣言であり，$x_1 : A_1, \ldots, x_n : A_n$ の形をしている．ここで，x_i は相異なる変数であり，A_i は，型または 種 である．

定義 7.2. (CC の 判定)

CC は，次の三つの 判定 を使う．

$$\vdash \Gamma \text{ context}$$

$$\Gamma \vdash K \text{ kind}$$

$$\Gamma \vdash t : A$$

ここで，Γ は 文脈，t は 項または型，A は型または 種，K は種 である． □

$\Gamma \vdash t : A$ は，Γ の変数宣言の下で項 t が型または 種 A に属することを表している．他の二つは補助的な 判定 である．$\vdash \Gamma$ context は Γ が正しい 文脈 であることを表し，$\Gamma \vdash K$ kind は Γ の変数宣言の下で式 K が 種 であることを表す．

定義 7.3. (CC の推論規則)

$$\vdash \langle\rangle \text{ context}$$

$$\frac{\Gamma \vdash A : * \quad x \notin \mathrm{Dom}(\Gamma)}{\vdash \Gamma, x : A \text{ context}}$$

$$\frac{\Gamma \vdash A \text{ kind} \quad x \notin \mathrm{Dom}(\Gamma)}{\vdash \Gamma, x : A \text{ context}}$$

$$\frac{\vdash \Gamma \text{ context}}{\Gamma \vdash * \text{ kind}}$$

$$\frac{\Gamma, x : A \vdash B \text{ kind}}{\Gamma \vdash \Pi x : A.B \text{ kind}} \text{(A)}$$

$$\frac{\vdash \Gamma \text{ context} \quad x : A \in \Gamma}{\Gamma \vdash x : A} \text{(B)}$$

$$\frac{\Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi x : A.B : *} \text{(C)}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A.M) : (\Pi x : A.B)} \text{(D)}$$

$$\frac{\Gamma \vdash M : (\Pi x : A.B) \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]} \text{(E)}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A' \,\mathrm{kind} \quad A =_\beta A'}{\Gamma \vdash M : A'}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A' : * \quad A =_\beta A'}{\Gamma \vdash M : A'}$$

ここで，$x \notin \mathrm{Dom}(\Gamma)$ は，Γ が，$x_1 : A_1, \ldots, x_n : A_n$ の形をしていて，$x_i \neq x \quad (1 \leq i \leq n)$ であることを表す．$x : A \in \Gamma$ は，Γ が，

$$x_1 : A_1, \ldots, x_n : A_n, x : A, y_1 : B_1, \ldots, y_m : B_m$$

の形をしていることを表す．□

次のことに注意しよう．規則 (A)(B)(C) 中の A は，$A : *$ である場合と A kind の場合がある．両方の場合を一つの規則で表している．規則 (D)(E) の A,B は，$A : *$ または A kind，および，$B : *$ または B kind の 4 通りの場合を一つの規則で表している．

定理 7.4.

1. $\Gamma \vdash M : A$，$\Gamma \vdash M : A'$ なら，$A =_\beta A'$.
2. $\Gamma \vdash M : A$，$M \xrightarrow[\beta]{*} M'$ なら，$\gamma \vdash M' : A$.
3. $\Gamma \vdash M : A$ なら，M は強正規化可能.
4. $\Gamma \vdash M : A$ なら，M は Church-Rossor の性質を満たす.
5. 判定 は決定可能である.

□

3．は，Girard による巧妙な方法により，型理論 F について最初に証明された [18]．CC でも，同様の方法を使って，3. が証明できる．

CC は強力なプログラミングシステムになっている．プログラムは CC の型付 λ 式 t で表す．1. により，項 t に対して型は，β 変換を除いて，一意的に定まる．2. により，t の β 簡約は t の型を保存する．3. により，型 A を持つ項 t は，どのような戦略で β 簡約を行っても，必ず停止し，正規形が得られる．4. より，t の計算結果である簡約により得られる正規形は，簡約の戦略によらず一意的に定まる．5. により，項 t と型 A が与えられたとき，$\vdash t : A$ であるか否かは，計算により有限時間で必ず判定できる．以上のことから，CC を，項 をプログラムとし，β 簡約を計算とするプログラミングシステムとして使うことができる．

CC では，項 を型で抽象することができる．この機能により，CC の型付けは多相的 である．例えば，一般的恒等関数 $\lambda\alpha : *.\lambda x : \alpha.x$ は，CC において，型 $\Pi\alpha : *.\Pi x : \alpha.\alpha$ を持つ．このことは次のように証明できる．

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\vdash \langle\rangle \text{ context}}{\vdash * \text{ kind}}}{\vdash \alpha : * \text{ context}}}{\alpha : * \vdash \alpha : *}}{\vdash \alpha : *, x : \alpha \text{ context}}}{\alpha : *, x : \alpha \vdash x : \alpha}}{\alpha : * \vdash (\lambda x : \alpha.x) : (\Pi x : \alpha.\alpha)}}{\vdash (\lambda\alpha : *.\lambda x : \alpha.x) : (\Pi\alpha : *.\Pi x : \alpha.\alpha)}$$

CC では，型上を動く変数を Π によって束縛することにより，高階論理で通常用いられる強力な方法を使って，論理記号や再帰的に定義される型を定義することができる．

一階述語論理の論理記号は，次のように定義できる．ここで，$A : *$, $B : *$, $C : A \to *$ とする．

$$\Sigma x : A.C(x) \equiv \Pi P : *.(\Pi x : A.C(x) \to P) \to P,$$

$$A \,\&\, B \equiv \Pi P : *.(A \to B \to P) \to P,$$

$$A \vee B \equiv \Pi P : *.(A \to P) \to (B \to P) \to P,$$

$$A \to B \equiv \Pi x : A.B,$$

$$\forall x : A.C(x) \equiv \Pi x : A.C(x),$$

$$\exists x : A.C(x) \equiv \Sigma x : A.C(x),$$

$$\bot \equiv \Pi P : *.P.$$

このように定義された論理記号に関して通常の一階述語論理の推論規則が CC 中で導出できる．

ライプニッツの $=$ は，次のように定義できる．ここで，$a : A, b : A$ とする．

$$a = b \equiv \forall P : A \to *.P(a) \to P(b)$$

CC では，型によって高階述語論理の論理式が表現できるので，型を用いてプログラムの仕様を強力に表現することができる．型付 λ 式で表されたプログラム t が型 A で表現された仕様を満たすことは，

$$t : A$$

として表せる．この方法の場合，プログラムの正当性の検証は型検査により実現でき

る.

7.2 再帰型の表現

プログラムの仕様を型または論理式で記述する場合，プログラムは再帰的なデータを扱い，また，プログラムは再帰的に繰り返されることにより制御されるため，再帰的性質が自然に記述できることが重要である [27, 43]. CC では，型上を動く変数を Π で束縛できることを用いて，再帰的に型を定義する機能が実現できる．再帰的に定義された型はその再帰的定義に対応する帰納法の原理を満たす.

自然数を表す述語 N は次の性質を満たす.

$$N(x) \leftrightarrow x = 0 \vee \exists y(N(y) \,\&\, x = \mathrm{succ}(y))$$

N は未知述語 P を含む次の方程式の最小解になっている.

$$P(x) \leftrightarrow x = 0 \vee \exists y(P(y) \,\&\, x = \mathrm{succ}(y))$$

最小性から，数学的帰納法の原理が導かれる．一般に，述語 $P(x)$ を定義する場合に $P(x)$ の定義を自然に書き表してゆくと $P(x)$ の定義本体の中に $P(x)$ 自体を再帰的に呼び出して使ってしまう場合がある．この場合，定義本体を $B(P)(x)$ で表すと，定義したい $P(x)$ は，

$$P(x) \leftrightarrow B(P)(x) \tag{7.1}$$

が成り立つ述語である．さらに，$P(x)$ により条件付けられるデータ x に関して帰納法の原理が使いたい．このため，定義したい述語 $P(x)$ は (7.1) を満たす最小の述語である．この条件により，$P(x)$ は存在すれば同値を除いて一意的に定まる.

$\mu(x)$ が $B(P)(x)$ により P に関して再帰的に定義された述語であるとは，次が成り立つことをいう.

$$\forall x : A.B(\mu)(x) \to \mu(x), \tag{μ1}$$

$$\forall C : A \to *.(\forall x : A.B(C)(x) \to C(x)) \to (\forall x : A.\mu(x) \to C(x)). \tag{μ2}$$

ただし，$A : *, \mu : A \to *, B : (A \to *) \to A \to *$ である.

定義本体 $B(P)(x)$ によって述語 $P(x)$ を再帰的に定義する場合，$P(x)$ が存在するためには，通常では $B(P)(x)$ に条件が必要である．この条件として，$B(P)(x)$ 中に，P が strictly positive にだけ出現する，または，positive にのみ出現する，または，$B(P)(x)$ が P に関して単調であるなどの条件が通常では必要となる.

CC では，再帰的定義の本体 $B(P)(x)$ が再帰的に定義される型 P に関して単調であるならば，$B(P)(x)$ により P に関して再帰的に定義された型 μ を次のように構成することができる．

$$\mu \equiv \lambda x : A.\forall P : A \to *.(\forall x : A.B(P)(x) \to P(x)) \to P(x).$$

ここで，単調とは，次が成り立つことである．

$$\forall P : A \to *.\forall Q : A \to *.(\forall x : A.P(x) \to Q(x)) \to$$
$$(\forall x : A.B(P)(x) \to B(Q)(x)).$$

ここで，

$$\mathrm{r}_1 \equiv \lambda x : A.\lambda a : B(\mu)(x).\lambda P : A \to *.\lambda b : (\forall x : A.B(P)(x) \to P(x)).$$
$$bx(m\mu P(\lambda x : A.\lambda y : \mu(x).yPb)xa),$$
$$\mathrm{r}_2 \equiv \lambda C : A \to *.\lambda a : (\forall x : A.B(C)(x) \to C(x)).\lambda x : A.\lambda b : \mu(x).bCa$$

とすると，μ に関して次が成り立つ．

$$\mu : A \to *,$$
$$\mathrm{r}_1 : \forall x : A.B(\mu)(x) \to \mu(x), \qquad (\mu 1)$$
$$\mathrm{r}_2 : \forall C : A \to *.(\forall x : A.B(C)(x) \to C(x)) \to (\forall x : A.\mu(x) \to C(x)). \qquad (\mu 2)$$

ただし，

$$A : *,$$
$$B : (A \to *) \to A \to *,$$
$$m : \forall P : A \to *.\forall Q : A \to *.(\forall x : A.P(x) \to Q(x)) \to$$
$$(\forall x : A.B(P)(x) \to B(Q)(x))$$

である．(μ1)は，$\mu(x)$ を μ の再帰的な定義にしたがって展開して得られる $B(\mu)(x)$ について，$B(\mu)(x)$ が成り立てば $\mu(x)$ が成り立つことを意味している．μ は，述語 $P : A \to *$ に関する条件

$$B(P)(x) \to P(x) \qquad (7.2)$$

を満たしている．(μ2)は，(7.2) が成り立つ述語 $P : A \to *$ の中で μ が最小であることを示している．

(μ1)(μ2)より，次が成り立つ．

$$(\lambda x : A.\lambda b : \mu(x).b(B(\mu))(m(B(\mu))\mu \mathrm{r}_1)) : \forall x : A.\mu(x) \rightarrow B(\mu)(x) \qquad (\mu 3)$$

ゆえに，

$$B(\mu)(x) \leftrightarrow \mu(x) \qquad (7.3)$$

であることがわかる．述語 $P : A \rightarrow *$ に関する条件

$$B(P)(x) \leftrightarrow P(x) \qquad (7.4)$$

は，未知述語 P を含む方程式と考えることができる．(7.3) より，μ は (7.4) の解である．(μ2)は，μ が (7.4) の最小解であることを表している．(μ1)(μ2)の 項 r_1, r_2 は，このような原理が CC で証明できることを示している．

再帰的に定義される型 μ をデータの型 $\mu : *$ であるものに限って考えてみよう．$B(P) : *$ は P に関して単調であると仮定する．

$$\mu \equiv \forall P : *.(B(P) \rightarrow P) \rightarrow P,$$
$$(\lambda a : B(\mu).\lambda P : *.\lambda b : B(P) \rightarrow P.b(m\mu P(\lambda y : \mu.yPb)a)) : B(\mu) \rightarrow \mu,$$
$$(\lambda C : *.\lambda a : B(C) \rightarrow C.\lambda b : \mu.bCa) : \forall C : *.(B(C) \rightarrow C) \rightarrow \mu \rightarrow C.$$

ここで

$$m : \forall P : *.\forall Q : *.(P \rightarrow Q) \rightarrow B(P) \rightarrow B(Q)$$

である．ここで，さらに，簡単のため特別な場合として未知集合 P に関する方程式

$$\begin{aligned} P &= \sum_{i=1}^{n} \prod_{j=1}^{k_i} T_j^i \\ &= T_1^1 \times \cdots \times T_{k_1}^1 + \cdots + T_1^n \times \cdots \times T_{k_n}^n \end{aligned} \qquad (7.5)$$

の最小解となる集合を表す型 $P : *$ を考えよう．ここで，$+$ は 直和，$\times$ は 直積 を表す．また，T_j^i 中に P は positive にのみ出現すると仮定する．(7.5) の最小解の型 T は次で与えられる．

$$T \equiv \forall P : *.S_1 \rightarrow \cdots \rightarrow S_n \rightarrow P,$$
$$S_i \equiv T_1^i \rightarrow \cdots \rightarrow T_{k_i}^i \rightarrow P. \qquad (1 \leq i \leq n)$$

7.3 再帰的データ型の表現例

自然数全体からなる型 Nat の例を考えよう．Nat は CC 中で再帰的に定義できる．型 Nat は方程式

$$\mathrm{Nat} = 1 + \mathrm{Nat}$$

の最小解である．ここで，1 は一点集合 である．ゆえに，上の方法より，

$$\mathrm{Nat} \equiv \forall P : *.P \to (P \to P) \to P$$

と定義できる．Nat を変形して，

$$\mathrm{Nat}' \equiv \forall P : *.(P \to P) \to P \to P$$

と定義しても，Nat′ も最小解を表す型である．Nat′ は，generic Church numeral の型になっている．Nat の要素を列挙してみよう．

$$\begin{aligned}
\overline{0} &\equiv \lambda P : *.\lambda x : P.\lambda y : P \to P.x, \\
\overline{1} &\equiv \lambda P : *.\lambda x : P.\lambda y : P \to P.yx, \\
\overline{2} &\equiv \lambda P : *.\lambda x : P.\lambda y : P \to P.y(yx), \\
\overline{n} &\equiv \lambda P : *.\lambda x : P.\lambda y : P \to P.y^n(x).
\end{aligned}$$

ただし，n は自然数，$y^0(x) \equiv x, y^{n+1}(x) \equiv y(y^n(x))$ である．また，successor 関数が次のように定義できる．

$$\mathrm{succ} : \mathrm{Nat} \to \mathrm{Nat},$$

$$\mathrm{succ} : \lambda n : \mathrm{Nat}.\lambda P : *.\lambda x : P.\lambda y : P \to P.y(nPxy).$$

この Nat に関して，自然数に関する recursor が作れる．まず，自然数の iterator Itr を作る．

$$\mathrm{Itr} \equiv \lambda P : *.\lambda a : P.\lambda b : P \to P.\lambda n : \mathrm{Nat}.nPab.$$

$P : *, a : P, b : P \to P$ とすると，次の iterator としての性質が成り立つ．

$$\mathrm{Itr} : \forall P : *.P \to (P \to P) \to \mathrm{Nat} \to P,$$

$$\mathrm{Itr}\,Pab\overline{0} = a,$$

$$\mathrm{Itr}\,Pab\overline{n+1} = b(\mathrm{Itr}\,ab\overline{n}).$$

型の 直積 $\times : * \to * \to *$ を次のように定める. $A : *, B : *$ として,

$$A \times B \equiv A \,\&\, B,$$

$$\text{pair} \equiv \lambda A : *.\lambda B : *.\lambda a : A.\lambda b : B.\lambda P : *.\lambda c : A \to B \to P.cab,$$

$$\text{pair} : \forall A : *.\forall B : *.A \to B \to A \times B,$$

$$\pi_1 \equiv \lambda A : *.\lambda B : *.\lambda c : A \times B.cA(\lambda x : A.\lambda y : B.x),$$

$$\pi_1 : \forall A : *.\forall B : *.A \times B \to A,$$

$$\pi_2 \equiv \lambda A : *.\lambda B : *.\lambda c : A \times B.cB(\lambda x : A.\lambda y : B.y),$$

$$\pi_2 : \forall A : *.\forall B : *.A \times B \to B.$$

と定めると, 次が成り立つ.

$$\pi_1 AB(\text{pair}\, ABab) = a,$$

$$\pi_2 AB(\text{pair}\, ABab) = b.$$

自然数の recursor Rec は, 次のように定義できる.

$$\begin{aligned}\text{Rec} \equiv \lambda P : *.\lambda a : P.\lambda b : \text{Nat} \to P \to P.\lambda n : \text{Nat}.\\ \pi_2 \text{Nat} P(\text{Itr}(\text{Nat} \times P)(\text{pair}\,\text{Nat} P\overline{0}a)\\ (\lambda c : \text{Nat} \times P.\,\text{pair}\,\text{Nat}P(\text{succ}(\pi_1\text{Nat}Pc))(b(\pi_1\text{Nat}Pc)(\pi_2\text{Nat}Pc)))).\end{aligned}$$

次の recursor の満たすべき性質が成り立つ. $P : *$, $a : P$, $b : \text{Nat} \to P \to P$ とすると,

$$\text{Rec} : \forall P : *.P \to (\text{Nat} \to P \to P) \to \text{Nat} \to P,$$

$$\text{Rec}\, Pab\overline{0} = a,$$

$$\text{Rec}\, Pab\overline{n+1} = b\overline{n}(\text{Rec}\, Pab\overline{n})$$

が成り立つ.

階乗を計算する関数 fact : Nat → Nat は, 次のように定義できる.

$$\text{plus} \equiv \lambda n : \text{Nat}.\,\text{Itr}\,\text{Nat}\overline{0}\,\text{succ},$$

$$\text{plus} : \text{Nat} \to \text{Nat} \to \text{Nat},$$

$$\text{mult} \equiv \lambda n : \text{Nat}.\,\text{Itr}\,\text{Nat}\overline{0}(\text{plus}\, n),$$

$$\text{mult} : \text{Nat} \to \text{Nat} \to \text{Nat},$$

$$\text{fact} \equiv \text{Rec}\,\text{Nat}\overline{1}(\lambda n : \text{Nat}.\lambda x : \text{Nat}.\,\text{mult}\, nx),$$

$$\mathrm{fact} : \mathrm{Nat} \to \mathrm{Nat}.$$

リストの例を考えよう．型 $A : *$ が与えられたとき，型 A の要素からなるリスト全体の型 $\mathrm{List}(A)$ は再帰的に定義できる． $\mathrm{List}(A)$ は，方程式

$$\mathrm{List}(A) = 1 + A \times \mathrm{List}(A)$$

の最小解である．ゆえに，

$$\mathrm{List}(A) \equiv \forall P : *.P \to (A \to P \to P) \to P$$

と定義すればよい．ゆえに，

$$\mathrm{List} \equiv \lambda A : *.\forall P : *.P \to (A \to P \to P) \to P,$$

$$\mathrm{List} : * \to *$$

と定義すればよい． $\mathrm{List}(A)$ の要素を考えよう．

$$\mathrm{nil} \equiv \lambda P : *.\lambda x : P.\lambda y : A \to P \to P.x,$$

$$\mathrm{nil} : \mathrm{List}(A),$$

$$\mathrm{cons} \equiv \lambda a : A.\lambda b : \mathrm{List}(A).\lambda P : *.\lambda x : P.\lambda y : A \to P \to P.ya(bPxy),$$

$$\mathrm{cons} : A \to \mathrm{List}(A) \to \mathrm{List}(A).$$

$a_i : A$ について，

$$[\,] \equiv \mathrm{nil},$$

$$[a_1, \ldots, a_n] \equiv \mathrm{cons}(a_1)([a_2, \ldots, a_n])$$

と表記すると，次が成り立つ．

$$[\,] : \mathrm{List}(A),$$

$$[a_1] : \mathrm{List}(A),$$

$$[a_1, a_2] : \mathrm{List}(A),$$

$$[a_1, \ldots, a_n] : \mathrm{List}(A).$$

自然数の場合と同様にリストに関する recursor である Listrec を定義することができ， $B : *$, $b : B$, $c : \mathrm{List}(A) \to A \to B \to B$, $a : A$, $l : \mathrm{List}(A)$ とすると次が成り立つ．

$$\mathrm{Listrec} : \forall P : *.P \to (\mathrm{List}(A) \to A \to P \to P) \to \mathrm{List}(A) \to P,$$

$$\mathrm{Listrec}\, Pbc\, \mathrm{nil} = b,$$

$$\text{Listrec}\, Pbc(\text{cons}(a)(l)) = cla(\text{Listrec}\, bcl).$$

再帰的型を CC で表現する場合，次のような二つの問題点がある．第一に，再帰的に定義された型に関して意味上当然成立すると期待される帰納法が導出できない場合がある．上の Natを例に考えよう．この Natに関して次の形の帰納法が成立することは上に述べた通りである．

$$\forall C : *.(C \to (C \to C) \to C) \to \text{Nat} \to C.$$

ここで，Natが自然数を表すことから，次の形の帰納法も当然成立すると期待されるが，これは CC の中では導出できない．

$$\forall A : \text{Nat} \to *.A(\overline{0}) \to (\forall n : \text{Nat}.A(n) \to A(\text{succ}(n))) \to \forall n : \text{Nat}.A(n).$$

第二に，再帰的型は，CC のような型全体を動く変数を ∀ に用いるような強力な機構を使わなくても，もっと弱い型理論でも可能である．predicative な型理論を再帰型が使えるように拡張する試みが研究されている．

CC は単純で強力な型理論なので，CC の証明検証器を中心とするプログラミングシステムがいくつかつくられている．CC によるプログラム合成では，再帰的型を用いるときの関数の繰り返しの制御，および，∃ に関して存在する値の計算に不要な情報の計算を省略する点の効率化が重要である [38].

$\mathbf{ITT}_n$ や LF[20] は，F や CC と異なり，Π で用いる変数は，必ず型の要素の上を動き，型上を動くことはない．このため，型の意味が明確である．型の意味はその型を構成する型の意味により完全に定まる．また，型を集合論的に解釈することもできる．

これに対して，F や CC では，Π で用いる変数は型の上を動く場合がある．ゆえに，型の意味をその型を構成する型の意味だけから決定することはできず，型の意味を与えることが容易ではない．また，型を集合論的に解釈することはできない [40]. 圏論を用いて型を解釈して意味を与える 圏論的意味論が研究されている．

第 8 章

λ-cube

8.1 λ-cube の定義

様々な型理論が提案され研究されているが，これらの型理論のある種のものは Barendregt の λ-cube によってうまく分類し整理して説明することができる．

λ-cube について説明する．今まで紹介した $\mathbf{ITT}_n$, F, CC について，相互の関連を λ-cube を使って説明する．また，LF と Fω を λ-cube を用いて簡単に紹介する．

定義 8.1. (λ-cube)

λ-cube とは，八つの型理論 $\lambda\!\rightarrow$, $\lambda\mathbf{2}$, $\lambda\underline{\omega}$, $\lambda\omega$, $\lambda\mathbf{P}$, $\lambda\mathbf{P2}$, $\lambda\mathbf{P}\underline{\omega}$, $\lambda\mathbf{C}$ の総称である．この八つの型理論を図 8.1のように立方体の形に並べた立方体も λ-cube とよぶ． □

λ-cube の八つのそれぞれの型理論を定義する．型，項，判定と推論規則の大部分は，八つのすべての型理論に共通である．そこで，八つの型理論の定義では，まずこの共通した型，項，判定，推論規則の定義を与え，次に八つのそれぞれの型理論の定義で異なっている推論規則の定義を与える．

定義 8.2. (λ-cube の型と項)

項式 A を次のように定める．

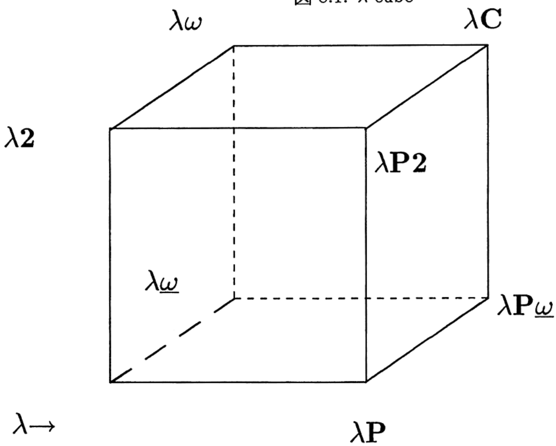
$$A ::= x | * | \Box | A_1 A_2 | \lambda x : A_1 . A_2 | \Pi x : A_1 . A_2$$

ここで，x は変数，$*, \Box$ はそれぞれ定数記号である．

λ-cube の型は，項式である．λ-cube の項は，項式である． □

λ-cube では CC と同様に 型全体の集まりである種を扱うことができる．$*$ は 型全体の集まりである種を表す定数記号である．$\Box$ は 種全体の集まりを表す定数記号である．

図 8.1: λ-cube



λ-cube の定義では，八つの型理論にできる限り一様な定義を与えることができるように，形式化を工夫する．このため 構文上は 項と型を区別しない．また，構文上は種は型に含まれる．しかし，意図している内容は CC の場合と同じものである．項と型と種の区別は 推論規則により与えられる．

定義 8.3. (λ-cube の判定)

λ-cube の文脈は

$$x_1 : A_1, \ldots, x_n : A_n$$

の形の式である．ここで，$n \geq 0$ であり，また，$1 \leq i \leq n$ について，x_i は変数，A_i は型である．λ-cube の判定は

$$\Gamma \vdash t : A$$

の形の式である．ここで Γ は文脈，t は項，A は型である．□

文脈 $x_1 : A_1, \ldots, x_n : A_n$ の意味は，$1 \leq i \leq n$ について各変数 x_i が型 A_i を持つことを表す変数宣言である．判定 $\Gamma \vdash t : A$ の意味は，文脈 Γ の表す変数宣言を仮定すると項 t が型 A を持つことが結論できるということである．

定義 8.4. (λ-cube の推論規則)

以下では，メタ変数 s, s_1, s_2 が $\{*, \Box\}$ 上を動くとする．また，Γ は文脈を，A,B

は型を， f,a は項を， x は変数を表す.

$$\vdash * : \square$$

$$\frac{\Gamma \vdash A : s \quad x \notin \mathrm{Dom}(\Gamma)}{\Gamma, x : A \vdash x : A} \text{(A)}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B : s \quad x \notin \mathrm{Dom}(\Gamma)}{\Gamma, x : B \vdash a : A} \text{(B)}$$

$$\frac{\Gamma \vdash f : (\Pi x : A.B) \quad \Gamma \vdash a : A}{\Gamma \vdash fa : B[x := a]}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B : s \quad A =_\beta B}{\Gamma \vdash a : B} \text{(C)}$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.B) : s_2} (s_1, s_2)\text{規則}$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash b : B \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\lambda x : A.b) : (\Pi x : A.B)} (s_1, s_2)\text{規則}$$

上で $x \notin \mathrm{Dom}(\Gamma)$ は， Γ が $x_1 : A_1, \ldots, x_n : A_n$ の形をしていて， $x \not\equiv x_i \quad (1 \leq i \leq n)$ であることを表している．最後の二つの規則を (s_1, s_2) 規則という．ここで $s_1, s_2 \in \{*, \square\}$ である.

上の推論規則 (A)(B)(C) は $s \equiv *$ の場合と $s \equiv \square$ の場合の二つの推論規則を 一つの推論規則で表している.

以上が 八つの型理論の推論規則の定義に共通した部分である.

(s_1, s_2) 規則に関しては， $s_1, s_2 \in \{*, \square\}$ である s_1， s_2 のどの組合せに対応する (s_1, s_2) 規則が型理論の推論規則に含まれるかは 八つの型理論によりそれぞれ異なる.

八つの型理論は それぞれ次に示す (s_1, s_2) に対応する (s_1, s_2) 規則を持つ.

$\lambda\!\rightarrow$	:	$(*, *)$			
$\lambda\mathbf{2}$	:	$(*, *)$	$(\square, *)$		
$\lambda\underline{\omega}$	:	$(*, *)$		$(\square, \square)$	
$\lambda\omega$	:	$(*, *)$	$(\square, *)$	$(\square, \square)$	
$\lambda\mathbf{P}$	:	$(*, *)$			$(*, \square)$
$\lambda\mathbf{P2}$	:	$(*, *)$	$(\square, *)$		$(*, \square)$
$\lambda\mathbf{P}\underline{\omega}$	:	$(*, *)$		$(\square, \square)$	$(*, \square)$
$\lambda\mathbf{C}$	:	$(*, *)$	$(\square, *)$	$(\square, \square)$	$(*, \square)$

□

8.2 λ-cube の性質

λ-cube では，上面の上にある型理論は (□,∗) を推論規則として持ち，右側面の上にある型理論は (∗,□) を推論規則として持ち，奥側面の上にある型理論は (□,□) を推論規則として持つ．

λ-cube 上の型理論は，立方体の辺に沿って上，右，奥の方向に行くにしたがって，推論規則が，(s_1, s_2) 規則の違いの分だけ増す．これらの型理論の構文は同じである．ゆえに，λ-cube 上のある型理論から 上，右，奥の方向に行くと，それを拡張した型理論に至る．拡張による順序付けに関して，最も小さい型理論が λ→ であり，最も大きい型理論が λC である．

λ-cube の八つの型理論のいくつかはこの解説中ですでに紹介した型理論と同じ型理論である．λ→ は 2 章で紹介した単純型理論 λ→ であり，λ2 は 6 章で紹介した F であり，λC は 7 章で紹介した CC である．ここで 二つの型理論が同じであるとは，それらが形式化が異なるため構文上は異なるように見えても 同じ内容を表していることである．正確にいうと，二つの型理論の間に型理論としての性質を保存する同型対応が存在することである．

λ-cube の八つのそれぞれの型理論は CC の場合と同様に次のようなよい性質を持つ．

定理 8.5.

1. $\Gamma \vdash M : A, \Gamma \vdash M : A'$ なら，$A =_\beta A'$.
2. $\Gamma \vdash M : A, M \xrightarrow[\beta]{*} M'$ なら，$\gamma \vdash M' : A$.
3. $\Gamma \vdash M : A$ なら，M は強正規化可能.
4. $\Gamma \vdash M : A$ なら，M は Church-Rossor の性質を満たす.
5. 判定 は決定可能である．

□

8.3 項と型の依存関係

(s_1, s_2) 規則の意味を説明しよう．この規則によって，項または型に依存する項または型を扱うことができる．

$(*,*)$ 規則について考えてみよう．$A : *$，$B : *$ とする．$(*,*)$ 規則の一つめの規則により 型 $\Pi x : A.B$ について $(\Pi x : A.B) : *$ であることが示せる．ゆえに 型 $\Pi x : A.B$ を持つ項を扱うことができる．また，$(*,*)$ 規則の二つめの規則により，$x : A$ と仮定すると $t : B$ ならば，型 $\Pi x : A.B$ を持つ項 $\lambda x : A.t$ を構成することができる．

$t : A, A : *$ のとき，A を型，t を型 A の要素という．$\lambda x : A.t$ は 要素 x に依存した要素 t を表している．$(*,*)$ 規則により 要素に依存した要素が扱える．

要素に依存した要素の例としては 通常の関数がある．例えば 型 A 上の恒等関数 $\lambda x : A.x$ は 要素に依存した要素を表している．

同様にして $(\Box,*)$ 規則により 型に依存した要素が扱える．型に依存した要素の例は多相的恒等関数

$$(\lambda\alpha : *.\lambda x : \alpha.x) : (\Pi\alpha : *.\alpha \to \alpha)$$

がある．

また，$(\Box,\Box)$ 規則により 型に依存した型が扱える．例えば

$$(\lambda\alpha : *.\alpha \to \alpha) : * \to *$$

は型に依存した型を表している．

また，$(*,\Box)$ 規則により 要素に依存した型が扱える．要素に依存した型の例は，A を型とすると，自然数 n を引数にとって型 $A^n \equiv A \times \cdots \times A$ を返す関数がある．また，述語論理における述語もこの規則により扱えるようになる．Curry-Howard の同型対応によって型と命題を同一視するとき，$P(a)$ が命題となる述語 P は，要素 a を引数にとって型 $P(a)$ を返す関数である．このような述語 P は $(*,\Box)$ 規則によって初めて扱える．

8.4 λ-cube と他の型理論との関係

λ-cube を用いることにより他の多くの型理論を整理して理解することができる．λ-cube の型理論は次のような他の型理論と同じ型理論である．

$\lambda\!\to$	:	単純型理論 $\lambda\!\to$
$\lambda\mathbf{2}$	:	F
$\lambda\omega$	:	Fω
$\lambda\mathbf{P}$	:	LF, ($\mathbf{ITT}_n$)
$\lambda\mathbf{C}$	:	CC

λ→ は明示的型付けである点を除いて単純型理論 λ→ (2 章) と同じである. **λ2** は Girard の F(6 章) と同じである. **λC** は CC(7 章) と同じである.

ITT$_n$ (5 章) は, 型であることと型が要素を持つことが相互再帰的に定義される点, 黙示的型付けである点, たくさんの組み込みの型や項を持つ点を除いて, **λP** と同じである.

型理論 LF, Fω は この解説中ではまだ紹介していない型理論である. 型理論 LF[20] は **λP** と同じ型理論であり, 型理論 Fω [14] は λω と同じ型理論である.

既知の型理論について, λ-cube により それらを同一の枠組の中でとらえることにより, それらの相互の関係の理解を深めることができる. また, 未知の型理論に対して λ-cube を使って 簡潔で完全な定義を与えることができる.

8.5 λ-cube の証明例

λ-cube の推論規則の理解のため λ-cube での証明の例を示す. ここでは, λ→ での **K** コンビネータの例 (2 章) と, F または CC での多相的恒等関数の例 (7 章) を, λ-cube において証明してみる. λ-cube と CC のそれぞれの形式化による推論規則が いずれも単純で強力なことを比較により理解するため, 同じ判定に対する CC での証明も併せて示す.

λ→ において **K** コンビネータの型付けを推論しよう. π_1 を次の証明図とする.

$$\dfrac{\dfrac{\dfrac{\vdash * : \Box}{A : * \vdash A : *} \quad \dfrac{\vdash * : \Box \quad \vdash * : \Box}{A : * \vdash * : \Box}}{A : *, B : * \vdash A : *}}{A : *, B : *, x : A \vdash x : A}$$

π_2 を次の証明図とする.

$$\dfrac{\dfrac{\dfrac{\vdash * : \Box \quad \vdash * : \Box}{A : * \vdash * : \Box}}{A : *, B : * \vdash B : *} \qquad \dfrac{\dfrac{\vdash * : \Box}{A : * \vdash A : *} \quad \dfrac{\vdash * : \Box \quad \vdash * : \Box}{A : * \vdash * : \Box}}{A : *, B : * \vdash A : *}}{A : *, B : *, x : A \vdash B : *}$$

次のようにして

$$A : *, B : * \vdash (\lambda x : A.\lambda y : B.x) : A \to B \to A$$

が証明できる.

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\,\pi_1 \\ A:*,B:*,x:A\vdash x:A\end{matrix}\quad\begin{matrix}\vdots\,\pi_2 \\ A:*,B:*,x:A\vdash B:*\end{matrix}}{A:*,B:*,x:A,y:B\vdash x:A}}{A:*,B:*,x:A\vdash(\lambda y:B.x):B\to A}}{A:*,B:*\vdash(\lambda x:A.\lambda y:B.x):A\to B\to A}$$

7 章で与えた CC の形式化によるとこのことは次のように証明される.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash\langle\rangle\text{ context}}{\vdash *\text{ kind}}}{\vdash A:*\text{ context}}}{A:*\vdash *\text{ kind}}}{\vdash A:*,B:*\text{ context}}}{A:*,B:*\vdash A:*}}{\vdash A:*,B:*,x:A\text{ context}}}{A:*,B:*,x:A\vdash B:*}}{\vdash A:*,B:*,x:A,y:B\text{ context}}}{A:*,B:*,x:A,y:B\vdash x:A}}{A:*,B:*,x:A\vdash(\lambda y:B.x):B\to A}}{A:*,B:*\vdash(\lambda x:A.\lambda y:B.x):A\to B\to A}$$

次に 一般的恒等関数の型付けを考えよう. これは 次のように λ2 で証明される.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash *:\Box}{\alpha:*\vdash\alpha:*}}{\alpha:*,x:\alpha\vdash x:\alpha}}{\alpha:*\vdash(\lambda x:\alpha.x):\alpha\to\alpha}}{\vdash(\lambda\alpha:*.\lambda x:\alpha.x):(\Pi\alpha:*.\alpha\to\alpha)}$$

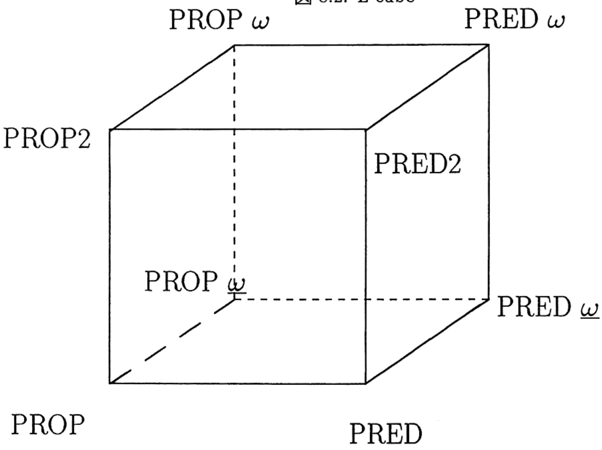
7 章に与えた CC の形式化によると同じ判定が次のように証明できる.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash *:\Box}{\vdash\alpha:*\text{ context}}}{\alpha:*\vdash\alpha:*}}{\vdash\alpha:*,x:\alpha\text{ context}}}{\alpha:*,x:\alpha\vdash x:\alpha}}{\alpha:*\vdash(\lambda x:\alpha.x):\alpha\to\alpha}}{\vdash(\lambda\alpha:*.\lambda x:\alpha.x):(\Pi\alpha:*.\alpha\to\alpha)}$$

8.6 L-cube

Curry-Howard の同型対応により λ-cube の八つの型理論にそれぞれ対応した八つの論理体系がある. これらの八つの論理体系も λ-cube と同様に 立方体により表現される包含関係を持つ. これが Barendregt の L-cube である [2].

図 8.2: L-cube



L-cube の一部の論理体系を紹介し，L-cube の性質を説明する．[2] では L-cube の論理体系が GTS の形で与えられているが，ここでは わかりやすさのため Church のソートを持つ論理体系の形で紹介する．

定義 8.6. (L-cube)

L-cube とは 八つの論理体系PROP， PROP2， PROP $\underline{\omega}$， PROP ω，PRED， PRED2， PRED $\underline{\omega}$， PRED ωの総称である．この八つの論理体系はそれぞれ次のようにもよばれる．

PROP	:	命題論理
PROP2	:	2 階命題論理
PROP $\underline{\omega}$	:	弱高階命題論理
PROP ω	:	高階命題論理
PRED	:	述語論理
PRED2	:	2 階述語論理
PRED $\underline{\omega}$	:	弱高階述語論理
PRED ω	:	高階述語論理

この八つの論理体系を図 8.2 のように立方体の形に並べた立方体も L-cube とよぶ．□

L-cube の八つの論理体系のうちPROP, PROP2, PRED ωの定義だけを以下では紹介する．他の五つの論理体系は λ-cube 上の型理論に同型に対応するようなこの三つの論理体系に似た形をした論理体系である．詳しくは [2] を参照していただきたい．

三つの論理体系を，ソートを持つ論理体系の形で定義する．ここで用いるソートを持つ論理体系は，一階述語論理と比べて，項がソートとよばれる型を持つ点，論理式は特定の型 Ω を持つ項である点が異なる．

三つの論理体系を，ソート，項，論理式，推論規則を与えることにより定義する．以下では 推論規則を自然演繹法の形で与える．また，項 M のソートを $[M]$ で表す．

定義 8.7. (PROP)

ソートは，Ω である．

項 M は次のように定義される．

$$M ::= c|M_1 \to M_2$$

ここで c は命題記号である．任意の項 M について $[M] = \Omega$ と定める．

論理式は $[M] = \Omega$ となる項 M であると定める．

推論規則は 次のようである．

$$\cfrac{\begin{matrix}[A]\\ \vdots \\ B\end{matrix}}{A \to B}\ (\to\mathrm{I}) \qquad\qquad \cfrac{A \to B \quad A}{B}\ (\to\mathrm{E})$$

ここで A,B は論理式である．□

L-cube の八つの論理体系を ソートを持つ論理体系という共通の枠組で与える理由は，λ-cube の八つの型理論と L-cube の八つの論理体系の間の八つの同型対応を 一つの枠組で論じるためである．

PROPは単純な型理論なので ソートや項を用いないで定義できる．ソートや項を用いない論理体系の形で表すと 次のようになる．

定義 8.8. (PROPの別の定義)

論理式 A は 次のように定義される．

$$A ::= c|A_1 \to A_2$$

ここで c は命題記号である．

推論規則は，定義 8.7 と同じである．□

PROP2も PROPと同様に ソートや項を使わないで定義できる. ここでは 簡単のため ソートや項を使わないで定義する. この定義を PROPの場合と同様に自明に拡張すれば ソートを持つ論理体系の形になる.

定義 8.9. (PROP2)

論理式 A は次のように定義される.

$$A ::= c|\alpha|A_1 \to A_2|\forall\alpha.A$$

ここで c は命題記号, α は命題変数を表す.

推論規則は, 次のようである.

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \to B}\ (\to\mathrm{I}) \qquad \frac{A \to B \quad A}{B}\ (\to\mathrm{E})$$

$$\frac{A}{\forall\alpha.A}\ (\forall\mathrm{I}) \qquad \frac{\forall\alpha.A}{A[\alpha := B]}\ (\forall\mathrm{E})$$

ただし (∀I) では A を導く推論の仮定中に α が現れてはならない. また, A,B は論理式, α は命題変数である. □

定義 8.10. (PRED ω)

ソート S は次のように定義される.

$$S ::= A|\Omega|S_1 \to S_2$$

ここで A は Ω 以外のソート記号である.

定数はそのソートと共に与えられているとする. また, 変数はそれぞれソートが定められているとする. 項 M とそのソート $[M]$ を 次のように定める.

1. 定数は 項である.
2. 変数は 項である.
3. M_1,M_2 が項, $[M_1] = [M_2] \to S$ なら, M_1M_2 は項であり, $[M_1M_2] = S$ である.
4. x が変数, $[x] = S$, M が項なら, $\lambda x \in S.M$ も項であり, $[\lambda x \in S.M] = S \to [M]$ である.
5. ϕ, ψ が項で, $[\phi] = [\psi] = \Omega$ なら, $\phi \to \psi$ も項であり, $[\phi \to \psi] = \Omega$ である.

6. x が変数，$[x]=S$，ϕ が項，$[\phi]=\Omega$ なら，$\forall x\in S.\phi$ は項であり，$[\forall x\in S.\phi]=\Omega$ である．

論理式 は $[\phi]=\Omega$ となる項 ϕ である．

推論規則は次のようである．

$$\frac{\begin{array}{c}[\phi]\\ \vdots\\ \psi\end{array}}{\phi\to\psi}\ (\to\mathrm{I}) \qquad \frac{\phi\to\psi\quad \phi}{\psi}\ (\to\mathrm{E})$$

$$\frac{\phi}{\forall x\in S.\phi}\ (\forall\mathrm{I}) \qquad \frac{\forall x\in S.\phi\quad [M]=S}{\phi[x:=M]}\ (\forall\mathrm{E})$$

ただし (∀I) では ϕ の推論の仮定中に x が自由に現れてはならない．また，ϕ,ψ は論理式，x は変数，S はソート，M は項である．□

λ-cube の型理論は Curry-Howard の同型対応によって それぞれ L-cube の論理体系に同型に対応する．$\lambda\to$ は PROPに，$\lambda 2$ は PROP2に，λC は PRED ωに，それぞれ同型に対応する．この同型対応の具体的な定義について詳しく知りたい場合は [2] を参照して考えていただきたい．

様々な形式化により 様々な内容を持ったたくさんの型理論が活発に構成され 研究されている．λ-cube により，これらの多くの型理論を一つの一般的な枠組によって形式化することができ，型理論の相互の関連を明確にすることができる．また，強正規化可能性や論理体系との同型対応などの型理論に関する一つの性質を，λ-cube 中のそれぞれの型理論に対して同時に考察することにより，その性質を深くとらえることができる．また，従来考えられていなかった型理論についても，すでに考えられているいくつかの型理論との関係を通して，その意義を明らかにすることができる．

λ-cube は，様々な形で活発に研究されている多くの型理論を，統一的な視点でとらえ，分類して整理して理解するために役立つ概念である．

第 9 章

Girard の逆説

9.1 Girard の逆説とは何か

型上を動く変数を束縛することにより F や CC のような強力な型理論が構成できた．強力な型理論は，その中で論理記号や再帰型が定義できるので，プログラミングシステムとしても有用である．しかし，この方向での型理論の拡張には限界がある．型上を動く変数を束縛する機能を広く許し過ぎると，強正規化可能性が成り立たなくなり，また，すべての型が要素を持つことになり，Curry-Howard の同型対応により対応する型理論が矛盾した論理体系になってしまうからである．この限界の一つが Girard の逆説によって与えられる．Girard の逆説とは，強力過ぎる型理論に対して Curry-Howard の同型対応により対応する論理体系が矛盾することを証明するための一つの証明方法である．

Girard の逆説は 集合論における Burali-Forti の逆説を型理論の中で行った証明方法である．Burali-Forti の逆説は，有名な Russel の逆説と同様に，素朴集合論における逆説である．

まず Russel の逆説を証明しよう．R を $x \notin x$ となる集合 x 全体の集まりとする．素朴集合論によれば R もまた集合である．ここで

$$x \in R \leftrightarrow x \notin x$$

が成り立つ．ここで $x := R$ とすれば

$$R \in R \leftrightarrow R \notin R$$

が成り立ち，矛盾する．

Burali-Forti の逆説は，Russel の逆説において R を順序数全体の集まりとして，ほぼ同様の方法で矛盾を導く．ここでは，以下に説明する Girard の逆説に近い形

で，R を well-founded な順序集合全体の集まりとして矛盾を導く方法を説明する．U を well-founded な順序集合全体の集まりとする．素朴集合論によれば U は集合である．U 上の順序 $<_U$ を，$a <_U b$ とは a が b に順序集合として埋め込めることと定めると，U は順序集合になる．ここで 任意の well-founded である順序集合が U に埋め込めることが示せる．さらに U は well-founded であることがわかる．ゆえに U は well-founded な順序集合であり，$U \in U$ となり，また，U は U に埋め込める．ゆえに $U <_U U$ である．これは U が well-founded であることに反し，矛盾する．

9.2 λU における Girard の逆説

Girard の逆説を型理論 λU において証明する．まず λU を定義する．

定義 9.1. (λU)

λU を λ-cube の記法で定義する．

定数は，$*, \square, \triangle$ である．

公理は，次のようである．

$$\vdash * : \square$$

$$\vdash \square : \triangle$$

推論規則として，次の (s_1, s_2) 規則を持つ．

$(*, *)$

$(\square, *)$

$(\square, \square)$

$(\triangle, \square)$

$(\triangle, *)$

□

λU では 種全体の集まりである超種 (super kind) が扱える．△ は 超種全体の集まりを表す定数である．λU は CC を 超種が扱えるように拡張した型理論である．

λU は 4 レベルを持つ CC[10] ともよばれる．CC は 要素，型，種の 3 レベルの構造を持つ．λU は 要素，型，種，超種の 4 レベルの構造を持つ．CC では，Girard の逆説が証明できず，また，強正規可能性が成り立つが，λU では，1 レベル上がったことにより，Girard の逆説が証明され，また，強正規化可能性も成り立たない．

以下では，型理論 λU と，これに Curry-Howard の同型対応により対応する論理体系を同一視する．対応する論理体系に対しても型理論 λU と同じ記法を用いる．対応する論理体系では $*$ が命題全体を表す．

逆説の証明で用いるいくつかの概念を定義する．

定義 9.2. (ordering)

$A : \square$, $R : A \to A \to *$, $d : A \to *$ のとき，(A, R, d) を ordering という． □

ordering は，順序集合に相当する．$\{x : A | d(x)\}$ が underlying set を表し，$R(x, y)$ がその上の順序 $x < y$ を表す．

定義 9.3. (埋め込み)

ordering (A, R, d), (B, S, e), $f : A \to B$, $b : B$ について次のように定める．

$$\begin{aligned}
&\mathrm{mor}(A, R, d, B, S, e, f, b) \equiv e(b) \mathbin{\&} (\forall x : A.d(x) \to e(f(x))) \mathbin{\&} \\
&\qquad (\forall x : A.d(x) \to \forall y : A.d(y) \to R(x, y) \to S(f(x), f(y))) \mathbin{\&} \\
&\qquad (\forall x : A.d(x) \to S(f(x), b)), \\
&\mathrm{mor} : \forall A : \square.(A \to A \to *) \to (A \to *) \to \\
&\qquad \forall B : \square.(B \to B \to *) \to (B \to *) \to \\
&\qquad (A \to B) \to B \to *, \\
&(A, R, d) < (B, S, e) \equiv \exists f : A \to B.\exists b : B.\mathrm{mor}(A, R, d, B, S, e, f, b), \\
&< : \forall A : \square.(A \to A \to *) \to (A \to *) \to \\
&\qquad \forall B : \square.(B \to B \to *) \to (B \to *) \to *.
\end{aligned}$$

□

$\mathrm{mor}(A, R, d, B, S, e, f, b)$ は，関数 f, 上限 b によって ordering (A, R, d) が ordering (B, S, e) に埋め込めることを表す．$(A, R, d) < (B, S, e)$ は，ordering (A, R, d) が ordering (B, S, e) に埋め込めることを表す．

定義 9.4. (推移性，well-foundedness)

ordering (A, R, d), $P : A \to *$ について 次のように定める．

$$\begin{aligned}
&\mathrm{trans}(A, R, d) \equiv \forall x : A.d(x) \to \forall y : A.d(y) \to \forall z.d(z) \to \\
&\qquad R(x, y) \to R(y, z) \to R(x, z), \\
&\mathrm{trans} : \forall A : \square.(A \to A \to *) \to (A \to *) \to *,
\end{aligned}$$

$$\mathrm{chain}(A,R,d,P) \equiv (\exists x : A.d(x)\ \&\ P(x))\ \&$$
$$(\forall x : A.d(x)\ \&\ P(x) \to \exists y : A.d(y)\ \&\ P(y)\ \&\ R(y,x)),$$
$$\mathrm{chain} : \forall A : \square.(A \to A \to *) \to (A \to *) \to (A \to *) \to *,$$
$$\mathrm{wf}(A,R,d) \equiv \forall P : A \to *.\neg\mathrm{chain}(A,R,d,P),$$
$$\mathrm{wf} : \forall A : \square.(A \to A \to *) \to (A \to *) \to *.$$

□

$\mathrm{trans}(A,R,d)$ は ordering (A,R,d) が推移的であることを表す． $\mathrm{chain}(A,R,d,P)$ は ordering (A,R,d) 中に P によって定まる部分集合中の要素からなる無限下降列が存在することを表す． $\mathrm{wf}(A,R,d)$ は ordering (A,R,d) が well-founded であることを表す．

定義 9.5. $((U, <_U, d_U))$

ordering (A,R,d) について 次のように定める．

$$U \equiv (\forall A : \square.(A \to A \to *) \to (A \to *) \to *) \to *,$$
$$U : \square,$$
$$i(A,R,d) \equiv \lambda x : (\forall A : \square.(A \to A \to *) \to (A \to *) \to *).x(A,R,d),$$
$$i : \forall A : \square.(A \to A \to *) \to (A \to *) \to U.$$

$u, v \in U$ について 次のように定める．

$$u <_U v \equiv \exists A : \square.\exists R : A \to A \to *.\exists d : A \to *.$$
$$\exists B : \square.\exists S : B \to B \to *.\exists e : B \to *.$$
$$i(A,R,d) = u\ \&\ i(B,S,e) = v\ \&\ (A,R,d) < (B,S,e),$$
$$<_U : U \to U \to *,$$
$$d_U(u) \equiv \exists A : \square.\exists R : A \to A \to *.\exists d : A \to *.$$
$$i(A,R,d) = u\ \&\ \mathrm{trans}(A,R,d)\ \&\ \mathrm{wf}(A,R,d),$$
$$d_U : U \to *.$$

□

U は すべての ordering の集まりに対応する． 関数 i によって ordering (A,R,d) を U 中に coding することにより U はすべての ordering の集まりであるという役割をはたす． $(U, <_U, d_U)$ 自身もまた ordering である．

命題 9.6. **((U,i) の性質)**

$$\begin{aligned}&\forall A:\Box.\forall R:(A\to A\to *).\forall d:(A\to *).\\&\quad\forall B:\Box.\forall S:(B\to B\to *).\forall e:(B\to *).\\&\quad i(A,R,d)=i(B,S,e)\to\\&\quad\forall P:(\forall A:\Box.(A\to A\to *)\to(A\to *)\to *).\\&\quad P(A,R,d)\to P(B,S,e).\end{aligned}$$

□

この命題は (U,i) が ordering を coding していることを表している. すなわち $i(A,R,d)=i(B,S,e)$ なら $A=B$, $R=S$, $d=e$ となる. 以下の証明では, (U,i) に関する性質として この命題 9.6 の性質だけを使い, (U,i) の定義自体は使わない.

命題 9.7. **($(U,<_U,d_U)$ の性質)**
(1) $\mathrm{trans}(U,<_U,d_U)$.
(2) $\mathrm{wf}(U,<_U,d_U)$.
□

証明 9.8.
(1) $x:U$, $d_U(x)$, $y:U$, $d_U(y)$, $z:U$, $d_U(z)$, $x<_U y$, $y<_U z$ を仮定する. $x<_U z$ を示す.

$d_U(x)$ より, $A_1:\Box$, $R_1:A_1\to A_1\to *$, $d_1:A_1\to *$ が存在して, $i(A_1,R_1,d_1)=x$, $\mathrm{trans}(A_1,R_1,d_1)$, $\mathrm{wf}(A_1,R_1,d_1)$ が成り立つ. 同様にして, $d_U(y)$, $d_U(z)$ より, $A_2:\Box$, $R_2:A_2\to A_2\to *$, $d_2:A_2\to *$, $A_3:\Box$, $R_3:A_3\to A_3\to *$, $d_3:A_3\to *$ が存在して, $i(A_2,R_2,d_2)=y$, $\mathrm{trans}(A_2,R_2,d_2)$, $\mathrm{wf}(A_2,R_2,d_2)$, $i(A_3,R_3,d_3)=z$, $\mathrm{trans}(A_3,R_3,d_3)$, $\mathrm{wf}(A_3,R_3,d_3)$ が成り立つ.

$x<_U y$ と 命題 9.6 より, $(A_1,R_1,d_1)<(A_2,R_2,d_2)$. ゆえに, $f_1:A_1\to A_2$, $a_2:A_2$ が存在して, $\mathrm{mor}(A_1,R_1,d_1,A_2,R_2,d_2,f_1,a_2)$ が成り立つ. 同様にして, $y<_U z$ と 命題 9.6 より, $f_2:A_2\to A_3$, $a_3:A_3$ が存在して, $\mathrm{mor}(A_2,R_2,d_2,A_3,R_3,d_3,f_2,a_3)$ が成り立つ. ゆえに $\mathrm{mor}(A_1,R_1,d_1,A_3,R_3,d_3,\lambda x:A_1.f_2(f_1(x)),f_2(a_2))$ が成り立つ. ゆえに $(A_1,R_1,d_1)<(A_3,R_3,d_3)$. ゆえに $x<_U z$. □

(2) $\mathrm{chain}(U, <_U, d_U, P)$ を仮定する．矛盾を示す．

$\mathrm{chain}(U, <_U, d_U, P)$ より，$u_0 : U$ が存在して，$d_U(u_0)$，$P(u_0)$ が成り立つ．$d_U(u_0)$ より，(A_0, R_0, d_0) が存在して，$u_0 = i(A_0, R_0, d_0)$，$\mathrm{trans}(A_0, R_0, d_0)$，$\mathrm{wf}(A_0, R_0, d_0)$ が成り立つ．

$$Q \equiv \lambda x : A_0.\exists A : \Box.\exists R : A \to A \to *.\exists d : A \to *.\exists f : A \to A_0.$$
$$d_U(i(A, R, d))\ \&\ P(i(A, R, d))\ \&$$
$$\mathrm{mor}(A, R, d, A_0, R_0, d_0, f, x),$$
$$Q : A_0 \to *$$

と定めると，$\mathrm{chain}(A_0, R_0, d_0, Q)$ が成り立つ．これは 次の 1, 2 からわかる．

1. $Q(a_0)$ かつ $d_0(a_0)$ となる $a_0 : A_0$ の存在を示す．

$u_0 : U$，$d_U(u_0)$，$P(u_0)$，$\mathrm{chain}(U, <_U, d_U, P)$ より，$u_1 : U$ が存在して，$d_U(u_1)$，$P(u_1)$，$u_1 <_U u_0$ が成り立つ．$d_U(u_1)$ より，(A_1, R_1, d_1) が存在して，$i(A_1, R_1, d_1) = u_1$．$u_1 <_U u_0$ と命題 9.6 より，$(A_1, R_1, d_1) < (A_0, R_0, d_0)$．ゆえに，$f_1 : A_1 \to A_0$，$a_0 : A_0$ が存在して，$\mathrm{mor}(A_1, R_1, d_1, A_0, R_0, d_0, f_1, a_0)$．ゆえに $Q(a_0)$ かつ $d_0(a_0)$ となる $a_0 : A_0$ が存在する．

2. $a : A_0, d_0(a), Q(a)$ を仮定する．$b : A_0, d_0(b), Q(b), R_0(b, a)$ となる b の存在を示す．

$Q(a)$ より，A_1, R_1, d_1, f_1 が存在して，$d_U(i(A_1, R_1, d_1))$，$P(i(A_1, R_1, d_1))$，$\mathrm{mor}(A_1, R_1, d_1, A_0, R_0, d_0, f_1, a)$ が成り立つ．

$\mathrm{chain}(U, <_U, d_U, P)$ より，$y : U$ が存在して，$d_U(y)$，$P(y)$，$y <_U i(A_1, R_1, d_1)$ が成り立つ．$d_U(y)$ より，(A_2, R_2, d_2) が存在して，$y = i(A_2, R_2, d_2)$，$\mathrm{trans}(A_2, R_2, d_2)$，$\mathrm{wf}(A_2, R_2, d_2)$ が成り立つ．命題 9.6 と $y <_U i(A_1, R_1, d_1)$ より，$(A_2, R_2, d_2) < (A_1, R_1, d_1)$．ゆえに，$f_2 : A_2 \to A_1$，$a_1 : A_1$ が存在して，$\mathrm{mor}(A_2, R_2, d_2, A_1, R_1, d_1, f_2, a_1)$ が成り立つ．ゆえに，$\mathrm{mor}(A_2, R_2, d_2, A_0, R_0, d_0, \lambda x : A_2.f_1(f_2(x)), f_1(a_1))$ が成り立つ．$b = f_1(a_1)$ とすればよい．

ゆえに $\mathrm{chain}(A_0, R_0, d_0, Q)$ が成り立つ．$\mathrm{wf}(A_0, R_0, d_0)$ であるから，矛盾する．□

命題 9.9.

ordering (A, R, d) について，$\mathrm{trans}(A, R, d)$，$\mathrm{wf}(A, R, d)$ なら $(A, R, d) < (U, <_U, d_U)$．□

証明 9.10.

$a : A$ について次のように定める.

$$d|_a \equiv \lambda x : A.d(x) \,\&\, R(x,a),$$

$$d|_a : A \to *,$$

$$f \equiv \lambda x : A.i(A,R,d|_x),$$

$$f : A \to U.$$

すると $\mathrm{mor}(A,R,d,U,<_U,d_U,f,i(A,R,d))$ が成り立つことが次の 1,2 からわかる.

1. $a : A$, $d(a)$, $b : A$, $d(b)$, $R(b,a)$ なら, $\mathrm{trans}(A,R,d)$ より, $\mathrm{mor}(A,R,d|_b,A,R,d|_a,\mathrm{id}_A,b)$ が成り立つ. ゆえに $i(A,R,d|_b) <_U i(A,R,d|_a)$ が成り立つ.

2. $a : A$, $d(a)$ なら, $\mathrm{mor}(A,R,d|_a,A,R,d,\mathrm{id}_A,a)$ が成り立つ. ゆえに $i(A,R,d|_a) <_U i(A,R,d)$ が成り立つ. □

命題 9.11. (Girard の逆説)

型理論 λ**U** は矛盾している. □

証明 9.12.

命題 9.7, 9.9 から $(U,<_U,d_U) < (U,<_U,d_U)$ が成り立つ. ゆえに

$$P \equiv \lambda x : U.x = i(U,<_U,d_U),$$

$$P : U \to *$$

と定めると $\mathrm{chain}(U,<_U,d_U,P)$ が成り立つ.

命題 9.7 (2) より $\mathrm{wf}(U,<_U,d_U)$ だから, 矛盾する. □

9.3 Girard の逆説が証明できる型理論

Girard の逆説が証明できる型理論について, Curry-Howard の同型対応により対応する論理体系は矛盾している. また, 型理論をプログラミングシステムとして役立てるために必要であったよい性質が失われ, 次のような悪い性質が成り立つ.

定理 9.13.

1. $\vdash t : (\forall \alpha : *.\alpha)$ となる項 t が存在する.
2. 任意の型 A に対して $\vdash t : A$ となる項 t が存在する.

3. $t : A$ であっても t が強正規化可能とは限らない.

4. $t : A, u : A$ のとき, $t =_\beta u$ は計算により決定可能ではない.

□

Girard の逆説は, CC の性質と命題 9.6 を満たす (U, i) の存在により導ける. (U, i) に関する条件は 命題 9.6 よりも次のように弱めることができる.

定理 9.14. ([10])

型理論 CC を拡張した型理論 T について, 型 U と関数 i が存在して次の条件が成り立つならば, T で Girard の逆説が証明できる.

- T での ordering の表現 $(A, R, d), (B, S, e)$ について,

  $$i(A, R, d) = i(B, S, e)$$

  ならば, (A, R, d) と (B, S, e) は順序同型である.

□

上の条件が成り立つとき (U, i) を universal system of notation for relations という.

Girard の逆説が証明できる他の型理論としては $\lambda*$ と強い直和を持つ CC がある.

定義 9.15. ($\lambda*$)

$\lambda*$ を, λ-cube の記法で定義する.

定数は $*$ である.

公理は 次のようである.

$$\vdash * : *$$

推論規則として 次の (s_1, s_2) 規則を持つ.

$$(*, *)$$

□

$\lambda*$ は Martin-Löf が $\mathbf{ITT}_n$ を構成する以前に構成した型理論として知られている [31]. この型理論は, $\lambda^{\tau\tau}$ [34] とも $\nu^{\tau\tau}$ [26] ともよばれる.

定理 9.16.

$\lambda*$ では Girard の逆説が証明できる. □

証明 9.17.

λU に対する証明で □,△ を $*$ で置き換えれば，$\lambda *$ における Girard の逆説の証明が得られる．□

Martin-Löf は 次のような自然な理由で $\lambda *$ を構成した．

1. Church の型理論の形式化によれば 命題全体は一つの型をなす．すなわち Prop : Type である．
2. Curry-Howard の同型対応により 命題と型は同一視できる．すなわち Prop = Type である．
3. ゆえに Type : Type である．

しかし，この型理論が矛盾していることが，上で説明した逆説を用いて，Girard により証明された．これを契機として Martin-Löf は predicative な型理論 $\mathbf{ITT}_n$ を構成するに至った．

定義 9.18. (強い直和を持つ CC)

λ-cube の記法を用いる．λC を拡張する方法によって強い直和を持つ CC を定義する．

項式 A を次のように拡張する．

$$A ::= x|*|\Box|A_1A_2|\lambda x : A_1.A_2|\Pi x : A_1.A_2|\Sigma x : A_1.A_2|(A_1, A_2)|\pi_1(A)|\pi_2(A)$$

強い直和を持つ CC の型と項は，項式である．

強い直和を持つ CC の推論規則は，λC の推論規則と，次の推論規則である．

$$\frac{\Gamma, x : A \vdash B : *}{\Gamma \vdash (\Sigma x : A.B) : *}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B[x := t]}{\Gamma \vdash (t, u) : (\Sigma x : A.B)}$$

$$\frac{\Gamma \vdash t : (\Sigma x : A.B)}{\Gamma \vdash \pi_1(t) : A}$$

$$\frac{\Gamma \vdash t : (\Sigma x : A.B)}{\Gamma \vdash \pi_2(t) : B[x := \pi_1(t)]}$$

□

強い直和を比較によって理解するため，弱い直和を定義する．弱い直和を持つ CC は，上の推論規則の最後の二つを次の推論規則で置き換えた型理論である．

$$\frac{\Gamma \vdash t : (\Sigma x : A.B) \quad \Gamma \vdash C : * \quad \Gamma \vdash u : (\Pi x : A.B \to C)}{\Gamma \vdash \mathrm{rep}(t,u) : C}$$

弱い直和は CC の中で 次のように構成できる．

$$\Sigma x : A.B \equiv \Pi P : *.(\Pi x : A.B \to P) \to P,$$

$$\mathrm{rep}(t,u) \equiv tCu.$$

ゆえに，弱い直和を持つ CC では，強い直和を持つ CC と異なって，Girard の逆説は証明できず，また，強正規化可能性が成り立つ．

定理 9.19. ([10])

強い直和を持つ CC では Girard の逆説が証明できる． □

証明 9.20.

要素を持つ型 $S : *$ を一つ選ぶ．その要素 $s : S$ を一つ選ぶ． S,s の選び方の例は，例えば，

$$S \equiv \Pi\alpha : *.\alpha \to \alpha,$$

$$s \equiv \lambda\alpha : *.\lambda x : \alpha.x$$

とすればよい．

$$U \equiv \Sigma A : *.\Sigma R : (A \to A \to *).A \to \Sigma y : *.S,$$

$$U : *,$$

$$i(A,R,d) \equiv (A,(R,\lambda x : A.(d(x),s)))$$

と定義すれば， (U,i) は命題 9.6 を満たす． λU に対する Girard の逆説の証明中で，この (U,i) を用いれば，強い直和を持つ CC に対する Girard の逆説の証明が得られる． □

型上を動く変数の束縛を強力に拡張して，なおかつ Girard の逆説を回避して強正規化可能性などのよい性質を持つ型理論を構成する研究として， ECC [29]， λ**Z** [5] がある．

定義 9.21. (λ**Z**)

λ-cube の記法を用いる．

定数は, $*, \Box, \triangle$ である.
公理は, 次のようである.

$$\vdash * : \Box$$

$$\vdash \Box : \triangle$$

推論規則として次の (s_1, s_2) 規則を持つ.

$(*, *)$

$(\Box, *)$

$(\triangle, *)$

$(*, \Box)$

$(\Box, \Box)$

$(*, \triangle)$

$(\Box, \triangle)$

$(\triangle, \triangle)$

$\Box$

定理 9.22. ([5])
$\lambda\mathbf{Z}$ では強正規化可能性が成り立つ. $\Box$

$\lambda\mathbf{Z}$ は, 型, 種の他に超種を持ち, $(\triangle, \Box)$ 以外のすべての (s_1, s_2) 規則を持つ. $\lambda\mathbf{Z}$ は CC を強力に拡張した型理論であるにもかかわらず, $\lambda\mathbf{Z}$ では Girard の逆説は成り立たない. このことから $\lambda\mathbf{U}$ における Girard の逆説の証明には $(\triangle, \Box)$ 規則が本質的に重要な役割を果たしていることがわかる. $\lambda\mathbf{U}$ における Girard の逆説の証明では $(\triangle, \Box)$ 規則が定義 9.5 および命題 9.6 で使われている.

第 10 章

文献紹介

この解説に関連のある型理論の勉強をこれから始めたい人のために 文献を紹介する．型理論に関する文献は多数あり，また，最近でも 活発に研究がなされているため文献も増え続けている．筆者の知っている範囲内で，入手が容易であるものにできるだけ限って，わかりやすく内容の整った最近の文献を紹介する．

計算機科学に型理論がどのように役立つかについてもっと詳しく知りたい人のためには，型理論を活用した実際のプログラミングシステムに関する文献として，まとまった本として [7, 37] がある．

型理論自体について理解を深めたい人には，はじめに次のような教科書や解説論文をたくさん読むことをお薦めする．

型理論の教科書としては，F に関する話をまとめたものとして [18] がある．また，$\mathbf{ITT}_n$ の教科書として [33] がある．また，構成的論理の教科書 [4] には $\mathbf{ITT}_n$ の説明がある．また，型なしラムダ計算の教科書 [24] には単純型理論 $\lambda\to$ に関する詳しい説明がある．

その他の教科書風にわかりやすく型理論に関して書かれたものとしては，[3, 21, 22, 23, 28, 41, 42] がある． [23] は計算機科学向けの数理論理学の教科書であり，型理論についても少し説明してある．その他は 解説論文である．

型理論で扱う型付ラムダ計算は 型なしラムダ計算に基づく．型なしラムダ計算にはよい教科書 [1, 24] がある．

次に特定の話題に関する文献を紹介する．

3 章の ML の型理論については，[12] がある． 3 章では [6] を紹介したが，[12] は標準的であり よくまとまってもいる．

4 章の型推論のアルゴリズムについては，[35] が最初のものである．記法が現在普通に用いられているものと大幅に異なっていて読みにくいが この解説はこれによって

いる．

5章の $\mathbf{ITT}_n$ は，[33] が教科書であり原典でもある．また，[4] でも解説されている．理解の助けになる関連した論文に [32] がある．また，$\mathbf{ITT}_n$ に基づいたプログラミングシステムに関する教科書 [7, 37] も理解の助けになるだろう．

6章の F は，[18] が教科書である．ただしこの教科書の意味論の部分を読むためには [16] が必要だろう．[14, 15] が原典である．

7章の CC は，[11] がまとまっている．[9] は，CC を道具として用いて様々な話題が CC 上で展開されている．CC を拡張する方向の研究の文献としては例えば [29] がある．

8章の λ-cube に関しては，[2] がまとまっている．この解説もこれによっている．

9章の Girard の逆説に関しては，[10, 26] をお薦めする．この解説もこれらによっている．[15] が原典である．

おわりに

最近の型理論の分野における評価の固まりつつある比較的完成された内容を持つことがらについて，型理論は計算機科学の役に立つという観点に重点をおいて，解説した．このために，基本的で代表的な五つの型理論 $\lambda\rightarrow$, **ML** の型理論，$\mathbf{ITT}_n$, F, CC を，具体的に紹介することにより，概説した．また，これらの型理論を整理して理解するために，λ-cube を紹介した．最後に，Girard の逆説について説明した．

ここで紹介した話題はすべて 最近では評価の定着したものばかりである．研究が現在進行中である様々な型理論を紹介できればもっと面白かったかも知れないが，紹介できなくて残念である．また，評価の定着した型理論の中でも $F\omega$, LF のような面白い型理論が詳しく説明できなかった．

この解説では，計算機科学に直接関係することがらを紹介するため，型理論の構文や形式化に重点をおいて説明した．このため，型理論の意味論に関しては，直観的な説明を与えるだけにし，ほとんど説明しなかった．しかし，型理論の意味論に関する研究も重要な話題である．

この解説が，型理論に関する基本的概念，性質や，現在話題になっている型理論の姿について，読者の具体的な理解の助けになることを筆者は願っている．

型理論は，それ自体の理論的興味も大きいが，計算機科学に役立つという点でも興味が尽きない．

参考文献

[1] Barendregt, H.P. : *The Lambda Calculus: Its Syntax and Semantics* (North-Holland, Amsterdam, 1984).

[2] Barendregt, H.P. : Introduction to Generalised Type Systems, *Proceedings of 3rd International Conference on Theoretical Computer Science, 1989* (World Scientific Publishing, Singapore).

[3] Barendregt, H.P. and Hemerik, K. : Types in Lambda Calculi and Programming Languages, *Proceedings of 3rd European Symposium on Programming, Lecture Notes in Computer Science* **432** (1990) 1–35.

[4] Beeson, M. : *Foundations of Constructive Mathematics* (Springer, 1985).

[5] Berardi, S.A. : A Subsystem λZ of Coquand's λU without Girard's paradox, Manuscript, 1989.

[6] Clément, D. : A Simple Applicative Language: Mini-ML, *Proceedings of ACM Conference on Lisp and Functional Programming* (1986) 13–27.

[7] Constable, R.L. et al : *Implementing Mathematics with the Nuprl Proof Development System* (Prentice-Hall, 1986).

[8] Coquand, T. and Huet, G. : A Selected Bibliography on Constructive Mathematics, Intuitionistic Type Theory and Higher Order Deduction, *J. Symbolic Computation* **1** (1985) 323–328.

[9] Coquand, T. and Huet, G. : A Higher Order Proof System of Mechanizing Mathematics, in: *EUROCAL 85, Lecture Notes in Computer Science 205* (1985) 152–184.

[10] Coquand, T. : An Analysis of Girard's Paradox, *Proceedings of Symposium on Logic in Computer Science* (1986) 227–236.

[11] Coquand, T. and Huet, G. : The Calculus of Constructions, *Information and Computation* **76** (1988) 95–120.

[12] Damas, L. and Milner, R. : Principal type-schemes for functional programs, *Proceedings of ACM Symposium on Principles of Programming Languages* (1982) 207–212.

[13] Fujita, K. et al : Relationship Between Logic and Type System, ソフトウェア科学会全国大会 (1989) 49–52.

[14] Girard, J.Y. : Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types, in: *J.E. Fenstad, ed., Proceedings of 2nd Scandinavian Logic Symposium* (North-Holland, 1971) 63–92.

[15] Girard, J.Y. : *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieure* (Thèse d'Etat, Université Paris VII, 1972).

[16] Girard, J.Y. : The System F of Variable Types, Fifteen Years Later, *Theoretical Computer Science* **45** (1986) 159–192.

[17] Girard, J.Y. : Normal Functors, Power Series and λ-calculus, *Annals of Pure and Applied Logic* **37** (1988) 129–177.

[18] Girard, J.Y., Lafont, Y. and Taylor, P. : *Proofs and Types* (Cambridge University Press, 1989).

[19] Gordon, M.J., Milner, A.J. and Wadsworth, C.P. : Edinburgh LCF, *Lecture Notes in Computer Science 78* (1979).

[20] Harper, R., Honsell, F. and Plotkin, G. : A Framework for Defining Logics, *Proceedings of Symposium on Logic in Computer Science* (1987) 194–204.

[21] 林 晋, 関数型言語における型理論, 情報処理 **29**(8) (1988) 845–852.

[22] 林 晋, 文献案内「型理論」, コンピュータソフトウェア **5**(1) (1988) 98–102.

[23] 林 晋 : 数理論理学、 コロナ社、 1989.

[24] Hindley, J.R. and Seldin, J. : *Introduction to Combinators and λ-Calculus* (Cambridge University Press, 1986).

[25] Howard, W.A. : The Formulae-as-types Notion of Constructions, in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* (Academic Press, 1980) 479–490.

[26] Howe, D.J. : The Computational Behaviour of Girard's Paradox, *Proceedings of Symposium on Logic in Computer Science* (1987) 205–214.

[27] 小林 聡, 龍田 真：直観主義的帰納的定義に対する実現可能性解釈, ソフトウェア科学会全国大会 (1989) 113–116.

[28] 小林 聡：構成的証明からのプログラム抽出, コンピュータソフトウェア, **7**(4) (1990) 3–18.

[29] Luo, Z. : ECC, an Extended Calculus of Constructions, *Proceedings of Symposium on Logic in Computer Science* (1989) 386–395.

[30] MacQueen, D., Plotkin, G. and Sethi, R. : An Ideal Model for Recursive Polymorphic Types, *Information and Control* **71** (1986) 95–130.

[31] Martin-Löf, P. : A Theory of Types, Unpublished Manuscript, 1971.

[32] Martin-Löf, P. : Constructive mathematics and computer programming, in: L.J. Cohen et. al, eds., *Logic, Methodology, and Philosophy of Science VI* (North-Holland, Amsterdam, 1982) 153–179.

[33] Martin-Löf, P. : *Intuitionistic Type Theory* (Bibliopolis, Napoli, 1984).

[34] Meyer, A.R. and Reinhold, M.B. : 'Type' is not a Type, *Proceedings of ACM Symposium on Principles of Programming Languages* (1986) 287–295.

[35] Milner, R. : A Theory of Type Polymorphism in Programming, *Journal of Computer and System Sciences* **17** (1978) 348–375.

[36] Mitchell, J.C. : Type Systems for Programming Languages, in: *Handbook of Theoretical Computer Science* **B** (Elsevier Science Publishers, 1990) 367 – 458.

[37] Nordström, B., Petersson, K. and Smith, J.M. : *Programming in Martin-Löf's Type Theory* (Oxford University Press, 1990).

[38] Paulin-Mohring, C. : Extracting Fω's Programs from Proofs in the Calculus of Constructions, *Proceedings of ACM Symposium on Principles of Programming Languages* (1989) 89–104.

[39] Reynolds, J.C. : Towards a Theory of Type Structure, *Lecture Notes in Computer Science* **19** (1974) 408–425.

[40] Reynolds, J.C. : Polymorphism is not Set-Theoretic, *Lecture Notes in Computer Science* **173** (1984) 145–156.

[41] Reynolds, J.C. : Three Approaches to Type Structures, *Lecture Notes in Computer Science* **185** (1985) 97–138.

[42] 桜井 貴文, 直観主義論理と型理論, 情報処理 **30**(6) (1989) 626–634.

[43] Tatsuta, M. : Program Synthesis Using Realizability, *Theoretical Computer Science* **90** (1991) 309–353.

[44] 龍田 真 : 型理論 I, コンピュータソフトェア **8**(1) (1991) 25–33.

[45] 龍田 真 : 型理論 II, コンピュータソフトェア **8** (2) (1991) 40–46.

[46] 龍田 真 : 型理論 III, コンピュータソフトェア **8** (3) (1991) 2–8.

[47] 龍田 真 : 型理論 IV, コンピュータソフトェア **8** (4) (1991) 56–68.

[48] Troelstra, A.S. : On the Syntax of Martin-Löf's Type Theories, *Theoretical Computer Science* **51** (1987) 1–26.

[49] *Standard ML Reference Manual (preliminary)*, 1989.

著者略歴

龍 田 真 (たつた まこと)
1960年 生まれ
1983年 東京大学法学部卒業
1985年 東京大学理学部情報科学科卒業
現 在 東北大学電気通信研究所助手
専 攻 プログラム理論，型理論

レクチャーノート／ソフトウェア学 1
型理論

1992年 11月 20日 初版発行

著 者 龍 田 真
発行者 櫻 井 慧 子
発行所 株式会社近代科学社
〒162 東京都新宿区市ケ谷田町2-7-15
定価 1300円 電話03(3260)6161 振替(東京)6-7625

中央印刷・松島製本 ISBN 4-7649-0201-X

本書は Amazon にて印刷・製本された、

オンデマンドペーパーバックです。

一部奥付に記載されている情報と異なる場合がございます。

1334914R00057

Produced by Amazon
Printed in Japan
落丁、乱丁本のお問い合わせは
Amazon.co.jp カスタマーサービスへ